Using Database Technology to Support Domain-Oriented Application Composition Systems

THESIS

Danny Alan Cecil
Captain, USAF

Joseph Alan Fullenkamp
Captain, USAF

AFIT/GCS/ENG/93D-03

AFIT/GCS/ENG/93D-03

Using Database Technology to Support Domain-Oriented Application Composition Systems

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Danny Alan Cecil, B.S.
Captain, USAF

Joseph Alan Fullenkamp, B.S.
Captain, USAF

December, 1993

21986

Acknowledgments

I would like to thank Major Paul Bailor, Maj Mark Roth, and Maj David Luginbuhl for their timely advice and thought-provoking questions during this thesis effort. Their guidance, confidence and encouragement were invaluable. I would also like to thank Mr. Dan Zambon and Mr. David Doak for providing a superb laboratory environment in which to conduct research. Those that provide outstanding service are quite often overlooked. I would like to thank the rest of the knowledge-based software engineering research group, especially my research partner Captain Joe Fullenkamp, for keeping me from going too far astray during this research.

Most of all I would like to thank the ones who made this all worthwhile. I would like to thank my parents, Doug and Jean Cecil, for instilling in me an understanding of the importance of family and a desire to always learn more. I would like to thank my daughters, Stacy Jean and Cristin Marie for understanding why I could not spend the time with them [illegible]ey deserved, and for the overwhelming joy and pride I get from being their dad. And finally, I would like to thank my wife Kim. Her understanding, support and unwavering love have sustained me through a lot in the past sixteen years. The sacrifices she made during this masters program are just the most recent in our Air Force career. It is to her I dedicate my work on this thesis.

Danny A. Cecil

I wish to thank my co-advisors, Major Mark Roth and Major Paul Bailor for their help, patience, and advice. Through their efforts, I learned alot about general problem research techniques as well as this specific research area. Thanks to Dr. Potoczny for being on my committee. I would also like to express my gratitude to other members of the knowledge-based software engineering group for all of their help and support. A special thanks to Captain Dan Cecil, my research partner, and Captain Jay Cossentine who helped me maintain some sense of sanity throughout the thesis process.

Finally, I would like to thank my family who supported me by understanding the importance of education. A special thanks goes to my daughters Erin, Katie, and Amy. Their understanding for all the time I needed to accomplish this task, as well as their ability to remind me of the important things in life helped me keep my priorities in line. And most of all, I want to thank my wife and best friend, Julie, for all the personal sacrifices she made for me. Her love, understanding, encouragement, and faith throughout this AFIT program, and all my endeavors, keeps me going through all life's difficulties and triumphs.

Joseph A. Fullenkamp

Table of Contents

List of Figures

AFIT/GCS/ENG/93D-03

Abstract

This research designed and prototyped an OODBMS technology base to store and retrieve various types of domain artifacts for domain-oriented application composition systems (DOACS). We developed object-oriented database schemas for a validating domain and the Object-Connection-Update software architecture. We implemented an inheritance relationship between the schemas so a domain model can inherit an architectural structure from an architecture model allowing us to isolate domain-specific knowledge from architecture-specific knowledge. We also developed a meta-model to formally define domain models in the database. We then developed a set of database methods to transform a domain model into a database schema for storing artifacts from the domain and to automatically populate the DOACS object base with the domain definition. Using an OODBMS, the structure and relationships that provide much of the power in object models are retained because the artificial flattening required for storage in traditional databases and file systems is prevented. Isolating domain and architecture models from each other has greatly increased the reusability of the domain artifacts, the domain model, and the architecture model. The inheritance relationship between domain and architecture models allows a domain to be defined once, but used in many different architectural environments and vice versa.

Using Database Technology to Support Domain-Oriented Application Composition Systems

I. Introduction

1.1 Overview

The primary purpose of this research effort was to examine the capabilities and benefits of using object-oriented database management system (OODBMS) technology in support of model-based software development (MBSD). This research focused on a prototype domain-oriented application composition system developed by previous research at AFIT (1, 26, 35), and explored the potential services an OODBMS could provide.

1.2 Background

"The [Joint Modeling and Simulation System] J-MASS program is intended to provide a flexible, standardized, and validated modeling tool to support a wide range of simulation requirements across the life cycle of a weapon system" (2). The major components of the J-MASS architecture, shown in Figure 1.1, are the Simulation Support Environment (SSE) and the Modeling Library. The SSE serves as the user interface to J-MASS, while the Modeling Library serves as a central repository for software simulation components to be used in J-MASS simulations. To date, research at the Air Force Institute of Technology (AFIT) has concentrated on the Develop Components and Assemble Players capabilities of the J-MASS SSE.

The Architect system, developed at AFIT by Randour (26), Anderson (1), and others, serves as a proof-of-concept for an advanced Develop Components and Assemble Players capability in the J-MASS SSE. Additionally, Anderson and Randour's work can be extended to provide rudimentary Configure and Execute Simulation capabilities for J-MASS. The shaded portion in the top half of Figure 1.2 shows the J-MASS capabilities that Architect currently provides. Note, too, that the bottom half of Figure 1.2 shows the

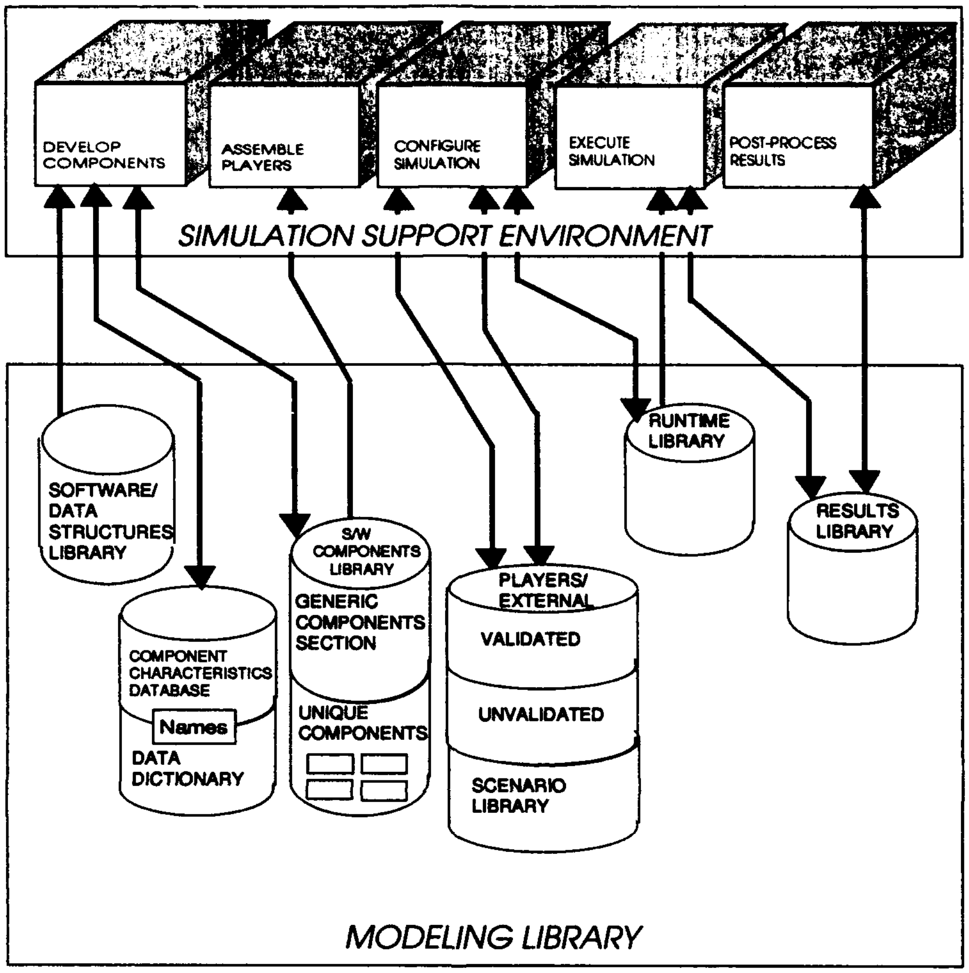


Figure 1.1 J-MASS Configuration : adapted from (2)

portion of the modeling library implemented using an OODBMS, and this is the primary focus of our research effort. Halloran's research (12) at AFIT investigates the use of an OODBMS for the remaining (dynamic) portion of the modeling library. Architect is a prototype system designed to allow a software engineer and application specialist to work together to compose formally specified software artifacts into formal software system specifications. Anderson and Randour used the logic circuits domain as their validating domain for Architect. They created primitive objects common to the logic circuits domain such as AND, OR, NAND, NOR, and NOT gates as well as switches and LEDs (light emitting diodes). Using these primitives and others, they demonstrated the ability to compose these artifacts into specifications of more complex artifacts in the logic circuits

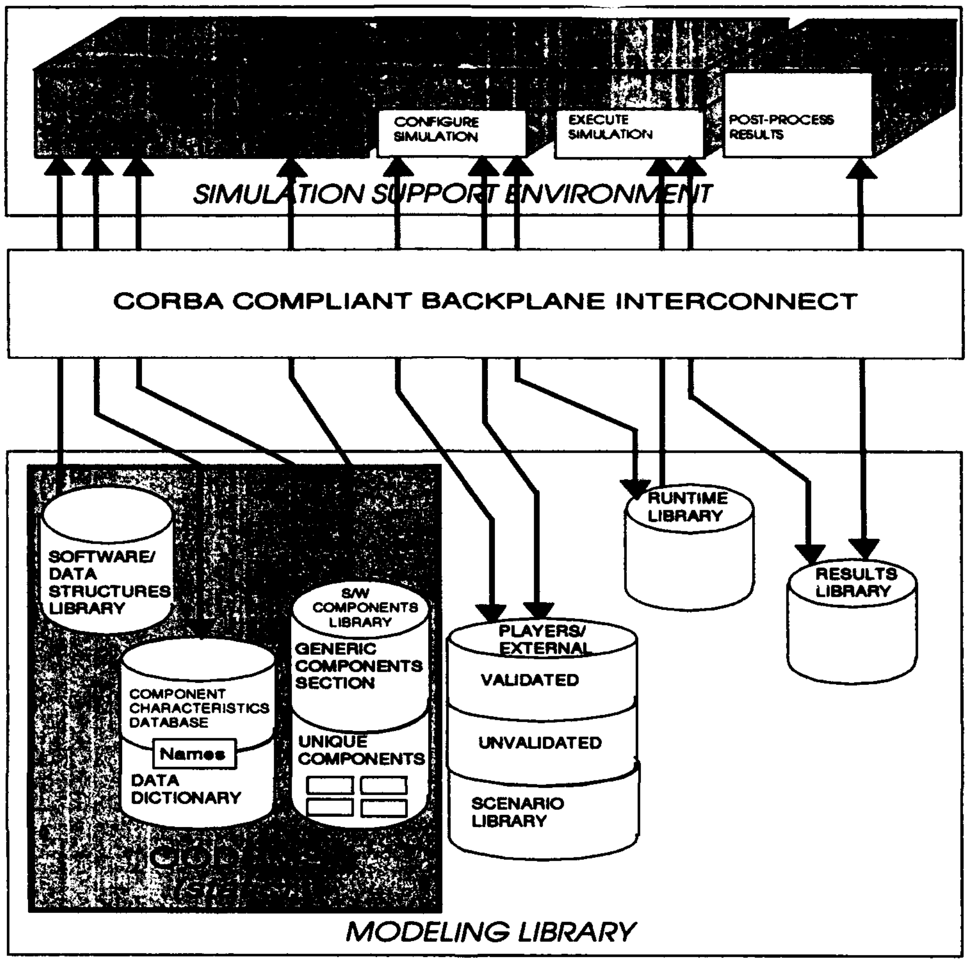


Figure 1.2 J-MASS/Architect Integration : adapted from (2)

domain such as half-adders, decoders, and binary-array multipliers. They also used the specifications of these artifacts to compose applications to test the functionality of the artifacts they had developed.

The primitives, components composed of primitives, and applications composed of primitives and other components are all examples of the types of artifacts that the technology base for the logic circuits domain must store. In the avionics domain, an artifact may be an aileron or a rudder and a system composed using those might be the airframe of an aircraft. The architectural model used by Randour and Anderson is the Object-Connect-Update (OCU) model developed at the Software Engineering Institute

(SEI) (19). The technology base needs to capture the chosen architecture as well as any domain models.

A primary objective listed in the J-MASS System Concept Document (SCD) is "to provide a Modeling Library which can be used to classify, catalog, and store software simulation components as reusable software"(2). A key practice in engineering is the extensive reuse of technical resources (10). The success of software engineering depends heavily on the reusability of software components and engineering models (knowledge), to include specifications, domain-specific models, source code, and documentation (4). To accomplish this, software engineers require a software library. A software library is a controlled collection of software resources and related documentation designed to aid in software development, reuse, and maintenance (25). Traditional database management systems (DBMSs) cannot fully support the complex, object-based artifacts that must populate the J-MASS Modeling Library (17:425-427).

A relatively new technology in the field of database management is OODBMS technology. Cattell states in his text that "the structural properties of complex objects alone necessitate the new ... [OODBMS] technology" (6:8). The complex, object-oriented nature of the artifacts to be stored in the J-MASS Modeling Library and the complimentary strengths of OODBMSs warrant an investigation of OODBMS technology for a possible implementation of the J-MASS Modeling Library.

Several factors drove the decision to use an OODBMS to implement portions of the J-MASS modeling library.

1. J-MASS has dictated that "the development approach [for simulation components] will be governed by ... an object-oriented requirements analysis and design [process]"(2:8).
2. The nature of the problem lends itself to an object-oriented solution.
    - (a) The simulation components are modeled as objects.
    - (b) Simulation objects can have very complex structures and be composed of other objects.

(c) There is a need for multiple versions of the database and of individual objects.

(d) According to Cattell (6:15), the three application areas most commonly cited as benefiting from OODBMSs are software engineering; mechanical and electrical engineering; and document systems. Our "application" deals with many of the same kinds of artifacts as those do. For example, we will be storing programs and program specifications as software engineering does, designs as mechanical and electrical engineering does, and text and graphics much like document systems do.

3. Three OODBMSs are available at AFIT for research efforts:

- ITASCA™
- OBJECTSTORE™
- MATISSE™

Boom and Mallare (21) demonstrated the feasibility of transforming informal specifications (such as entity relationship, state transition, and data flow models of domain objects) into formal, mathematically-based executable specifications amenable to manipulation and proof. The approach they took was first to translate the three informal models into a single Unified Abstract Model (UAM) which they developed. Then they mapped the UAM representation of a portion of the problem space to a formal Object Modeling Language (OML) they developed for that purpose. They used Software Refinery's REFINE™ to specify and execute their models. Because OML is formal in nature, Software Refinery's language processor, DIALECT™, can parse OML into the REFINE Object Base where it can be executed to simulate the behavior of the initial informal specifications. A user can make modifications until obtaining the expected behavior, thus producing a satisfactory formal specification. These formal object-based specifications can populate the Architect Technology Base and, or the J-MASS Modeling Library. They can then become the building blocks for composing together simulation components of use to J-MASS.

There are a number of other systems producing formal object-based specifications similar to those stored in the Architect technology base. Lockheed, for example, is doing work that may be integrated with the Architect system. Lockheed's Environment for

Automatic Programming (LEAP) produces object-based artifacts in the form of Common Intermediate Design Language (CIDL) primitives. Research is underway to provide a tool set necessary to introduce these specifications into the Architect system so the application specialist can manipulate them to produce composite systems (30).

During the life cycle of an application being developed with Architect, the application specialist can define objects at varying levels of abstraction. Actual target language code fragments; intermediate design languages like OML, CIDL, and REFINE; and informal graphical models are examples of different levels of abstraction that the application specialist can use. During any given Architect session, objects from a particular application domain, and other closely related domains, may populate the technology base. These objects can range in complexity from simple to very complex. During the prototyping of an application, many versions of objects may be developed. Many different instances of the technology base, representing many diverse domains, may need to be maintained between Architect sessions. The result is voluminous amounts of complex object-based models that need to be maintained.

OML, CIDL, REFINE, and Ada code are all abstract representations of complex objects that J-MASS could use and the J-MASS Modeling Library needs to maintain. To accomplish this, we must produce a common, object-based representation capable of modeling these complex objects. The schema of an OODBMS implementation can then use this common representation.

1.3 Problem

To date, research at AFIT has concentrated on the "develop components" and "assemble players" capabilities of the SSE, producing models that the modeling library needs to maintain. The next logical step in the J-MASS research effort is to develop a set of unified models to represent the data and objects described above. These models can help "bridge the gap" between Architect, LEAP, and other closely related systems; and an OODBMS can store these artifacts for the systems to share.

Problem Statement:

Use OODBMS technology to develop a Modeling Library that can classify, catalog, and store complex, object-oriented software simulation components as reusable software artifacts. The Modeling Library must be able to manage simulation artifacts produced and consumed by J-MASS, ARCHITECT, and LEAP.

1.4 Scope

Boom and Mallare (21) demonstrated the capability to transform informal analysis and design models into an object modeling language (OML), and then, in turn, convert that into a formal executable specification. Therefore, this research does not focus on the translation process of levels of abstraction. Instead, it provides a common repository for artifacts defined at various levels of abstraction by multiple sources that are accessible to many applications and users.

In this research we focused on the static portion of the J-MASS Modeling Library as depicted in Figure 1.2. We made no attempt to support the execution of an active simulation in the OODBMS environment, as this is the focus of other AFIT research (12).

There were three OODBMSs available at AFIT. They were ITASCA, MATISSE, and OBJECTSTORE. The developers of each of these OODBMSs added DBMS features to an existing object-oriented programming language. The other approach to OODBMS development is to add object-oriented capabilities to an existing traditional DBMS. We only considered the OODBMSs currently available at AFIT for our Modeling Library implementation.

1.5 Approach

The goal of this research was to implement a prototype of the static portion of the J-MASS Modeling Library using OODBMS technology. The steps needed to accomplish this goal follow:

- Conduct a literature search for parallel research efforts to aid in selecting the best OODBMS and developing a database schema for this problem.

- Identify the types of artifacts we want to store in the object-base and develop storage and retrieval tests to execute against the finished product.
- Analyze the artifacts and develop object model representations for them.
- Identify the characteristics needed in an OODBMS for storing the artifacts identified above.
- Select ITASCA™, OBJECTSTORE™, or MATISSE™ based on the above listed characteristics.
- Investigate ways to provide a standard communication interface that allows various tools in an domain-oriented application composition environment to use a database version of a technology base.
- Implement the technology base design in the chosen environment.
- Load the technology base with the artifacts identified above.
- Execute the test cases developed earlier to store, manipulate, and retrieve artifacts in the Modeling Library.

1.6 Summary

This research focused on a specific problem in a specific application area: managing complex, object-based artifacts for systems like J-MASS. We can generalize a solution to this specific problem, however, and apply it to a whole class of similar problems. Computer aided software engineering (CASE), computer aided design (CAD), hypertext, and office information systems (OIS) all manage complex objects in a dynamic environment. Traditional DBMSs are not meeting the needs of J-MASS and similar systems. Object-oriented database management system technology has the potential to meet these needs. This research helps identify the strengths and weaknesses of OODBMS technology for solving this class of problems.

1.7 Order of Presentation

The remainder of this thesis is organized as follows:

Chapter II provides a review of available literature concerning model-based software development and object-oriented database systems, and reasons why they should be combined.

Chapter III discusses the implementation of an application composition system we modified, the operational concept for developing technology bases using OODBMS technology, and our planned objectives.

Chapter IV examines our design and implementation methods used for this research effort.

In Chapter V we discuss our objectives for validating the correctness of our work, as well as reporting our test results.

Finally, Chapter VI summarizes our accomplishments and outlines recommendations for future research in this area.

II. Literature Review

2.1 Introduction

Before discussing what object-oriented database management systems (OODBMS) are and what benefits they hold for us, it is first necessary to define the application domain we worked with. We begin by defining some of the terminology needed to discuss domain-oriented application composition environments. We follow this with justification for using an object-oriented approach, and a description of the object modeling technique we used to describe the objects stored in the database. Finally, we investigate the evolution of OODBMS technology to its current state, and discuss its applicability to model-based software development.

2.2 Domain-Oriented Application Composition Terminology

Anderson (1) and Randour's (26) research focused on the development of a domain-oriented application composition system to help build up a technology base for software development. They discussed how this reusable technology base could help bring a more traditional engineering mind-set to the evolving software engineering discipline. Rather than duplicate their justifications and conclusions, we only provide definitions for some of the terminology we used as we developed an object-oriented database extension to their work.

The Software Engineering Institute published a special report on model based software development. They defined the following terms and the relationships between them as shown in Figure 2.1 (19):

- Technology Base – A set of models, and rules for their composition, designed and created with the same engineering goals in mind. That is, the engineering goals provide a common, underlying basis for the models. The technology base contains both product and practice models.
- Model – A scalable unit of reusable engineering experience or knowledge. A model is supported by specification forms and implementation templates which provide an indication of its structure, performance, and behavior. Models are constructed in the context of engineering goals.

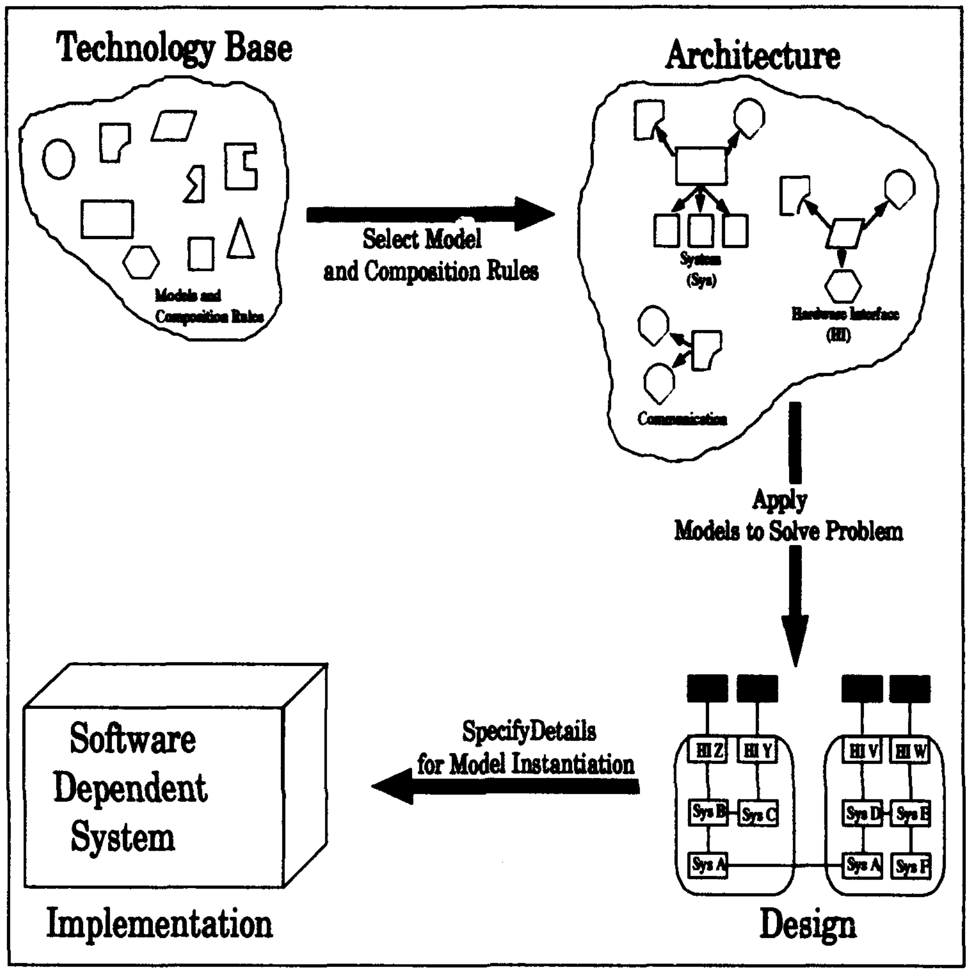

Technology Base
Architecture
Models and Composition Rules
Select Model and Composition Rules
System (Sys)
Hardware Interface (HI)
Communication
Apply Models to Solve Problem
HI Z
HI Y
HI V
HI W
Sys B
Sys C
Sys D
Sys E
Sys A
Sys A
Sys F
SpecifyDetails for Model Instantiation
Software Dependent System
Implementation
Design

Figure 2.1 Relationship of Model-Based Software Development Terms (adapted from (19))

- Engineering Goals – Overriding considerations in the design and construction of engineered products. For software engineering, goals may include: maintainability, enhanceability, composability, and so on.
- Architecture – A *style* of design. An architecture is a selection, from a technology base, of models and composition rules that defines the structure, performance, and use of a system relative to a set of engineering rules.
- Design – A composition resulting from the application of an architecture to describe a problem. The design has its structure and performance expressed by the model set comprising an architecture.
- Implementation – The software product (a specific instance of a design).

A Domain Engineer, highly knowledgeable in a specific domain, can capture knowledge of the domain in the form of domain models. These domain models can be combined with an architecture model to design and implement software products in a given target language. The reuse of models in the design and implementation stages is consistent with other classical engineering disciplines (10). Models can be abstracted to varying levels of detail. Anderson and Randour's work (1, 26) identified an approach that allows sophisticated end users, called Application Specialists, to compose applications within a given domain using pre-defined domain models. The approach Anderson and Randour describe is a domain-oriented application composition system. Objects and composition rules are maintained as executable, formal specifications in the object base, providing Software Engineers and Application Specialists with the ability to rapidly prototype and validate desired system behaviors without first having to build complete implementations (5). Randour and Anderson chose to use an object-oriented approach to capture abstractions of real-world entities of interest in the domain as well as those entities' behaviors and relationships.

2.3 Object-Oriented Approach and Modeling

An object-oriented approach revolves around a fundamental construct, the object, which combines both data structures (attributes) and behaviors (methods) in a single entity (29:1). The main benefit of object-oriented programming and design is not reduced development time. Object-oriented development may, in fact, take more time than conventional development, because it is intended to promote future reuse and reduce downstream errors and maintenance (29:9). Other benefits of an object-oriented approach are the ability to better model the world in which we live and work, to create models that include both physical and behavioral characteristics, and to protect data from corruption (7).

Object-oriented systems without an object storage mechanism flatten their object structures into records and use traditional storage and database managers. This translation process corrupts the object model and its associated benefits are lost (7). However, the definitions, integrity, and benefits of object models can be retained using object-oriented database management systems (7).

2.3.1 Definition of Terms. Before discussing object-oriented databases, it is first necessary to define some common definitions for databases, as well as terms and representations for object modeling. The following list is not comprehensive but contains terms and information we used throughout this document, and it is meant to define the framework in which we used them.

- Database Schema – The structure and integrity rules for application data stored in a database. This structure is the DBMS equivalent of the type definitions in a programming language (6).
- Meta-Model – A model that describes objects necessary to define models. For instance, we develop a meta-model to hold domain model definitions in the database.
- Object Model – The description of object structures in an object-oriented design –– their identity, their relationships to other objects, their attributes, and their operations (29). This is usually documented following some graphical form such as an object diagram. Note that for object-oriented databases, the object model and its object diagram representation describe the database schema.
- Object Diagram – a formal graphical notation for modeling objects, object classes, and their relationships to one another (29:23).
- Object Class – A description of a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects, and common semantics (29:22).
- Abstract Class – A class that is not directly instantiable but has instantiable descendant classes, each of which inherit the attributes, operations, and relationships of the parent abstract class(es) (29:61-62). These are normally used to define characteristics that will be inherited by more than one subclass. An abstract class must have at least one concrete subclass, that is, it cannot be the leaf of an inheritance tree.
- Concrete Class – A class that is instantiable. That is, it can have direct instances (29:61). A concrete class may have abstract subclasses (but they in turn must have concrete descendants). Only concrete classes can be leaf classes in an inheritance tree.
- Instance – A single instantiation of an object belonging to an object class. Each instance has its own attribute values and relationships to other instances of objects.
- Attribute – A pure data value (not an object) held by the objects in a class (i.e., *name, age,* and *weight* are attributes of *Person* objects, and *color, weight,* and *model-year* are attributes of *Car* objects) (29:23).

- Association – Describes a physical connection or a conceptual relationship between object classes. Often association names appear as verbs (29). For example, a person *works-for* a company.
- Multiplicity – Specifies how many instances of one class may relate to each instance of another class in an association.
- Ordering – Normally an association from a group of instances of one class to each instance of another class can be thought of as an unordered set. By specifying that an association is ordered, we signify that order is important for this relationship.
- Aggregation – The "*part-whole*" or "*a-part-of*" relationship in which objects representing *components* of something are associated with an object representing the entire *assembly* (29). This relationship is also referred to as an "*is-composed-of*" relationship.
- Inheritance – An abstraction for sharing similarities among classes while preserving their differences, it is the relationship between a class and one or more further refined versions of it. The class being refined is called the *superclass* and each refined version is called a *subclass* (29:38–39). For example, *Airplane* is the superclass of *F-16* and *B-52*. Each subclass *inherits* the features (attributes, operations, and relationships) of its superclass. For example, *B-52* inherits attributes *manufacturer, weight,* and *wing-span* from *Airplane*, as well as the operations *takeoff* and *land*. Sometimes called the "*is-a*" relationship because each instance of a subclass is an instance of the superclass as well (29).

2.3.2 Object Model Notation. Now that we have defined some of the terminology, we present a description of the notation used throughout this document for representing object models. We used a subset of Rumbaugh's Object Modeling Technique (OMT) (29).

Figure 2.2 provides the subset of OMT notation we used when developing our object models. Note that we represented objects as a rectangle containing the class name, and optionally the attributes and/or operations defined for the class. A special type of attribute is the class attribute. When a class attribute is inherited, the attribute definition, as well as the attribute value, is inherited. With a non-class attribute, only the definition is inherited and each object inheriting the attribute can have a unique value for that attribute. Rumbaugh distinguishes an attribute as a class attribute by placing a $ in front of the attribute name on the object model. Abstract classes are shaded to distinguish them from concrete classes.

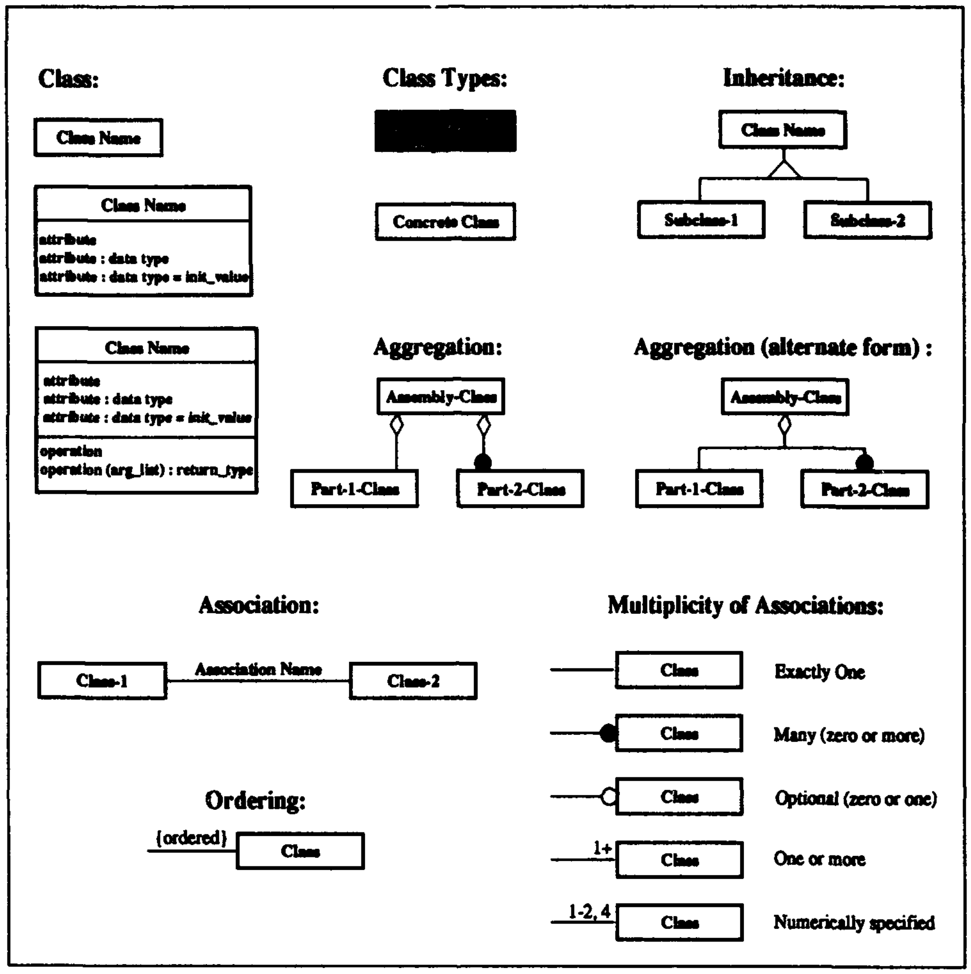


Figure 2.2 Object Modeling Notation (adapted from (29))

Now that we have described the application we will be working with and explained our terminology and representation method, we examine how object-oriented database technology can be applied to it.

2.4 Background on Object-Oriented Database Management Systems

Object-oriented database management systems (OODBMS) are a relatively new phenomenon in the software engineering community. As such, there is a lack of standards for OODBMSs to adhere to and a lack of benchmarks by which to judge them. Many decision-makers would like a clear picture of the potential benefits and drawbacks of using

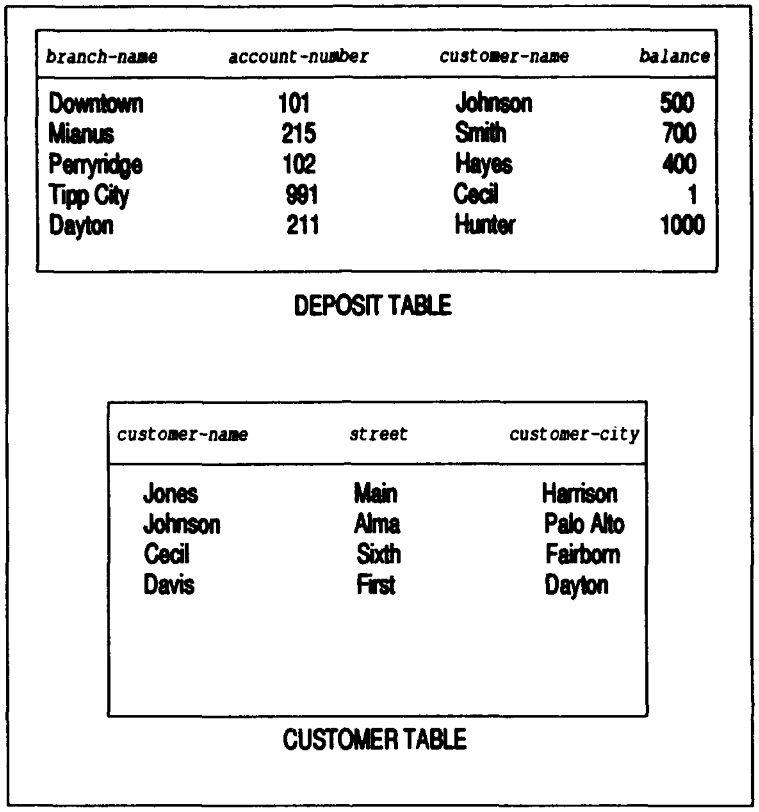
OODBMS technology. Our evaluation of the current state of OODBMS technology shows that there are two primary types of OODBMSs in use today. It further shows that certain classes of applications may benefit from one of these two types.

After explaining the motivation and introducing some key terms, we consider some of the major application areas driving the need for OODBMS technology. We then highlight some important characteristics of these applications to show how traditional DBMSs cannot satisfy their demands. A discussion of the direction of current research trends comes next, followed by an introduction to the two schools of thought on how OODBMS technology should evolve.

2.5 Evolution of Database Technology

A DBMS is a collection of related data and a set of procedures to access and manipulate that data (17:1). DBMSs are very beneficial when large volumes of data need to be managed (17:1). Traditionally, developers of DBMSs have sought to meet the needs of the business community. They typically deal with the type of data that simple data structures and operations can represent. For example, a bank might keep a record for each account holder that includes the account number, the member's name and social security number and the account balance. The operations performed on this data are relatively simple as well. When an account holder withdraws money from an account, an operation is invoked to reduce the account balance of the customer by the appropriate amount. This type of DBMS provides simple data management (6:2).

There have been a number of data models and architectures developed over the years to efficiently meet the needs of traditional database users. The most popular data model for commercial data processing applications today is the relational model (17:53). A relational database consists of a collection of tables of related data. Figure 2.3 depicts tables that build upon the banking example to illustrate how a relational database might store banking data. A row in a table represents a relationship between each of the columns in that row. For instance, from the deposit table we see that Johnson has an account numbered 101 at the Downtown branch and his balance is 500 dollars. Duplicating a value from one table in another table forms a relationship between the two tables. For example,



branch-name	account-number	customer-name	balance
Downtown	101	Johnson	500
Mianus	215	Smith	700
Perryridge	102	Hayes	400
Tipp City	991	Cecil	1
Dayton	211	Hunter	1000

DEPOSIT TABLE

customer-name	street	customer-city
Jones	Main	Harrison
Johnson	Alma	Palo Alto
Cecil	Sixth	Fairborn
Davis	First	Dayton

CUSTOMER TABLE

Figure 2.3 Relational Database Tables : adapted from (17:54,56)

by looking up Johnson's name in the customer table, one can see that he lives on Alma Street in the city of Palo Alto. A relational database management system (RDBMS) can manage large volumes of this simple type of data.

Some of the key characteristics of the more traditional databases are that they store large amounts of similarly structured records with small, simple data items. The length of a transaction on the traditional database is usually very short, measured in fractions of a second. Once designed and implemented, the structure of the database remains fairly static (17:425-426). The traditional DBMSs don't perform well, though, in many relatively new application areas including:

- Computer-aided design: A CAD database stores data pertaining to an engineering design, including the components of the item being designed, the interrelationship of components, and old versions of designs.
- Computer-aided software engineering: A CASE tool database stores data required to assist software developers. These data include source code, dependencies among software modules, definitions and uses of variables, and the development history of the software system.
- Multimedia databases: A multimedia database contains spatial data, audio data, video data, and the like. Databases of this sort arise from geophysical data, voice mail systems, and graphics applications.
- Office information systems: Office automation includes workstation-based tools for document creation and document retrieval, tools for maintaining appointment calendars, and so on. An OIS database must allow queries pertaining to schedules, documents, and contents of documents.
- Expert database systems: An expert database system includes not only data but also explicit rules representing integrity constraints, triggers, and other knowledge about the enterprise modeled by the database.

These application areas have made great gains in recent years, due in part to the increased capabilities and decreased cost of computer hardware. As their development evolved, so did the development of specialized database management systems to meet their special needs. The result has been the development of many stand-alone database systems that will not interface with any other application. With networked engineering workstations becoming more common, users and developers are looking for an application-independent DBMS capable of managing complex objects that many applications can share.

2.6 The Current State of Object Technology

A discussion of OODBMS technology and the applications driving its development is now in order. Something CASE, CAD, OIS, and expert systems all have in common are their object-oriented nature and hence their inherent need to manage complex objects. A complex object is a real world object one would consider to be a single object, but is composed of other objects. For example, a book is a complex object composed of a table of contents, some chapters, a glossary, and a bibliography. Any one of those objects can be further decomposed into a set of lower-level objects. For instance, a chapter could be composed of paragraphs, a paragraph of sentences, and so on. These relationships are very difficult and costly to represent in traditional DBMSs.

Executable code modules, called methods, are the implementation of operations on a class of objects (29). Since the data inside an object should only be accessible by calling these methods, we should store the methods together with the data objects.

Another major difference from RDBMSs is that OODBMSs need to support long-duration transactions. In a CAD or CASE environment, a designer may work on an object for hours before committing the work. RDBMSs cannot easily support this requirement.

There are four characteristics generally accepted as required for an object-oriented software approach (29:1-4):

1. Identity. Identity means discrete, distinguishable entities called objects represent data. Each object has its own identity independent of the values of its attributes.
2. Classification. A class is an abstraction for grouping objects with the same attributes and operations. Each object is an instance of a class.
3. Inheritance. The sharing among classes of attributes and operations in a hierarchical relationship. A program can break a class into subclasses, each of which inherits all the properties of its superclass.
4. Polymorphism. The same operation may behave differently on different classes. Each object knows how to perform its own operation. For example, the display of a circle is different from the display of a square. If a square and a circle are subclasses of a graphics superclass, then both have an operation called display, even though the implementation is different.

These are just a few of the important issues that the OODBMS field must address. For a more thorough discussion, see Cattell's text (6:15-44).

We might best describe the current state of the OODBMS field as a state of change. Existing OODBMSs are not based on a common data model and there is a lot of research and experimentation going on (28). Two schools of thought seem to be forming among the experts about the best way to develop OODBMSs (28). The authors of "The Third-Generation Database System Manifesto" advocate building object management capabilities into existing RDBMSs (31). The authors of "The Object-Oriented Database System Manifesto" on the other hand advocate not building on the limited RDBMS, but incorporating database management system capabilities into object-oriented programming languages (3). The first approach seems to show more promise for business applications, while the latter seems better suited for the new applications discussed in this chapter. In any event, it appears that both approaches will find a share of the DBMS market for some time to come (6:2).

2.6.1 Object-Oriented Database Programming Languages. In 1989, a group of experts published "The Object-Oriented Database System Manifesto" (3) to begin debate over the definition and theoretical framework for OODBMS. The consensus of this group was that the focus should be on staying consistent with the current object-oriented programming languages. Although there is a requirement for database management system features such as persistence of data, secondary storage management, concurrency, recovery, and ad hoc query capability, the addition of these features should not result in any loss of capability in the programming language. The convenience and efficiency of one language to be used for database and programming operations, and the current popularity of the object-oriented approach in programming are the strengths in this approach (6). The current lack of standards to allow portability of these systems across various platforms, and the minimal experience and development effort in these systems, compared to RDBMSs, are the weaknesses of this approach (6:233-235).

2.6.2 Extended Relational Database Management Systems. In response to the above mentioned approach (3), another group published "The Third-Generation Data

Base System Manifesto" (31) in 1990. The major theme of this group was that although relational databases couldn't handle the needs of complex applications, extensions could allow them to do so. Although most critics of the relational model point out the deficiencies in such applications as CASE and CAD, second generation systems do not support most business data processing applications all that well (31:2). The focus of this group, then, is that everybody requires a better DBMS, and there is, in fact, a surprising consensus on desired capabilities of this next generation of DBMSs. However, it is important to not throw out all the lessons learned and advantages offered by current systems. Some object-oriented programming features, such as inheritance and identity, are good ideas; however, the cost of these should not be the loss of reliability and data independence. DBMS access should only occur through a separate query language. Experience has shown that DBMSs should not allow user programs to navigate through the database in an ad hoc fashion (31:24). The natural evolution of existing relational DBMSs to ones with all the capabilities discussed in (31) is happening. The most important disagreement this group has with the object-oriented database programming language group is a matter of implementation, as the programming language approach currently violates many of the tenets that make good database systems.

2.7 Conclusion

Although OODBMSs lack the maturity, experience, and research that RDBMSs have, the success of object-oriented techniques in software analysis, design, and programming justifies consideration of OODBMSs when researching DBMS options. The field is progressing rapidly with considerable research involving OODBMSs and work has begun on establishing OODBMS standards (23, 24).

III. Operational Concepts and Requirements Identification

3.1 Overview

Research has been ongoing at the Air Force Institute of Technology (AFIT) to develop a domain-oriented application composition system. Anderson and Randour made great strides toward this end with their work on Architect (1, 26). Their requirements analysis includes the following quote from an article by Lowry (20) which clearly and succinctly sums up the philosophy and goals of domain-oriented application composition.

> Software engineering will evolve into a radically changed discipline. Software will become adaptive and self-configuring, enabling end users to specify, modify and maintain their own software within restricted contexts. Software engineers will deliver knowledge-based application generators rather than unmodifiable application programs. These generators will enable an end user to interactively specify requirements in domain-oriented terms ... and then automatically generate efficient code that implements these requirements. In essence, software engineers will deliver the knowledge for generating software rather than the software itself.
>
> Although end users will communicate with these software generators in domain-oriented terms, the foundation for the technology will be formal representations ... Formal languages will become the lingua franca, enabling knowledge-based components to be composed into larger systems. Formal specifications is the interface between interactive problem acquisition components and automatic program synthesis components.
>
> Software development will evolve from an art to a true engineering discipline. Software systems will no longer be developed by hand crafting large bodies of code. Rather, as in other engineering disciplines, components will be combined and specialized through a chain of value-added enhancements. The final specializations will be done by the end user. KBSE (knowledge-based software engineering) ... will not replace the human software engineer; rather, it will provide the means for leveraging human expertise and knowledge through automated reuse. New sub disciplines, such as domain analysis and design analysis, will emerge to formalize knowledge for use in KBSE components. (20:629-630)

AFIT researchers have already shown that "application composition systems, such as ... [those described by Lowry], are feasible" (1:7). They have also shown that "non-programmers, who are very knowledgeable about a particular domain, can create quite sophisticated applications without the direct assistance of software professionals" (1:7). This research effort and other research at AFIT (8, 11, 30, 33, 34, 36) seek to build on

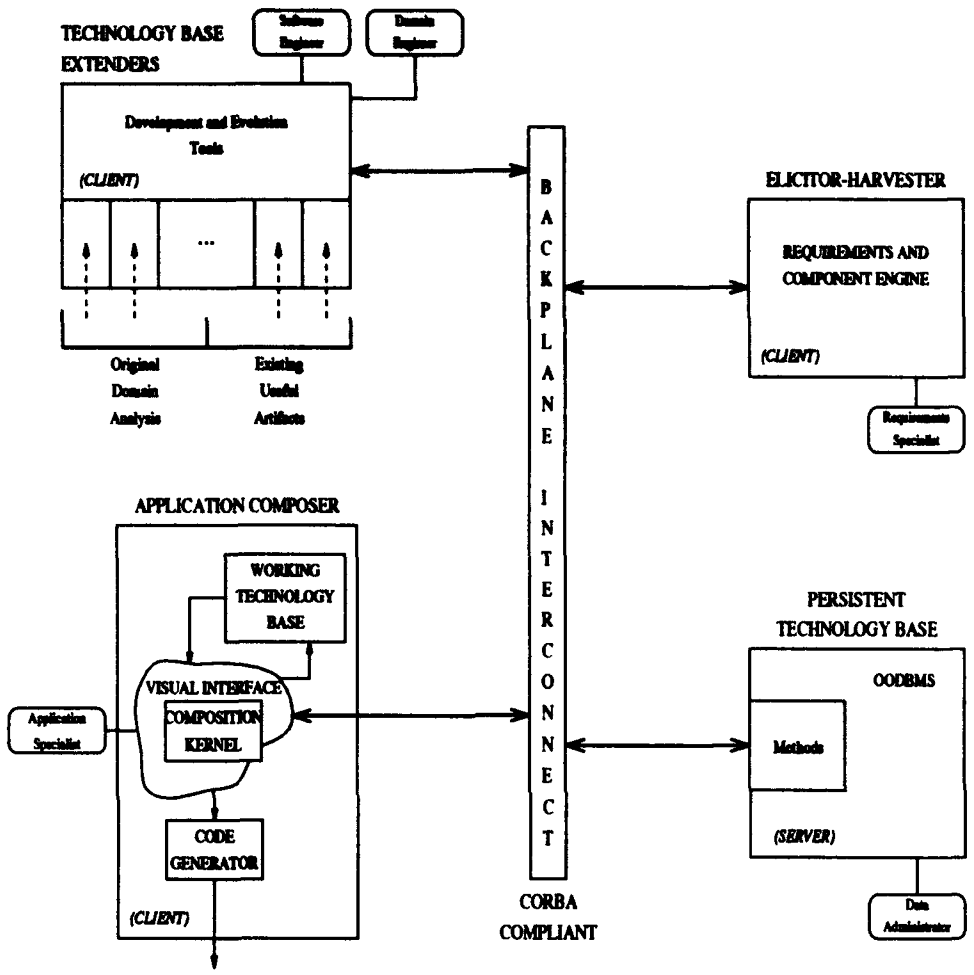


Figure 3.1 Domain-Oriented Application Composition Environment

the groundwork laid by Randour and Anderson, as well as Weide (35) who provided a visual system interface, called AVSI[1], to support the Architect domain-oriented application composition system.

Figure 3.1 depicts a domain-oriented application composition *environment*. It supports not only a domain-oriented application composition system like Architect, but also tools that can interact with Architect to create a more productive software engineering environment. Architect is a specific instance of the application composer pictured in the lower left-hand corner of the figure. An investigation of the Technology Base Extender

[1]The terms "Architect" and "AVSI" both refer to the baseline system we extended, and may be used interchangeably throughout this document

(upper left-hand corner of Figure 3.1) was conducted by Sandy (30) and Warner (34). The Elicitor-Harvester (upper right-hand corner of Figure 3.1) is the subject of future research.

The primary concern of this thesis effort is to provide an OODBMS implementation of the technology base shown in the lower right-hand corner of Figure 3.1 and to investigate the best means to provide a functional backplane interconnect shown in the center of the figure. To make such a domain-oriented application composition environment a reality, the technology base must manage large volumes of data representing knowledge about multiple domains and software architectures as well as software artifacts developed in the environment. If we expect to get maximum reuse of domains, architectures, and software artifacts developed in the domain-oriented application composition environment, we must have a well-defined standard interface, allowing other tools to access the environment easily.

This chapter addresses the issues involved in providing the services required of a technology base and how to make those services available to client tools in a domain-oriented application composition environment. As mentioned before, our work revolves around extending Architect version 1.0, so we begin in Section 3.2 by describing it and showing how it functioned as a stand-alone domain-oriented application composition system. In Section 3.3 we identify the requirements and goals we set for ourselves, based on our analysis of Architect 1.0 and our requirements analysis for a domain-oriented application composition environment. Finally, Section 3.4 covers an analysis of the object-oriented database management systems available to us at the start of our research, as well as the rationale for our selection.

3.2 Architect Operational Overview

Figure 3.2 is a system-level view of the Architect version 1.0 domain-oriented application composition system, and the environment in which it was used. To prepare Architect for use in a particular domain, an expert in that domain (Domain Engineer) performs a domain analysis to identify those components needed to build software applications in that domain. The Software Engineer, using knowledge of software architectures, formalizes this domain knowledge into a domain model with a set of primitive components. The Application Specialist can then start a session in Architect, at which time all the pre-defined

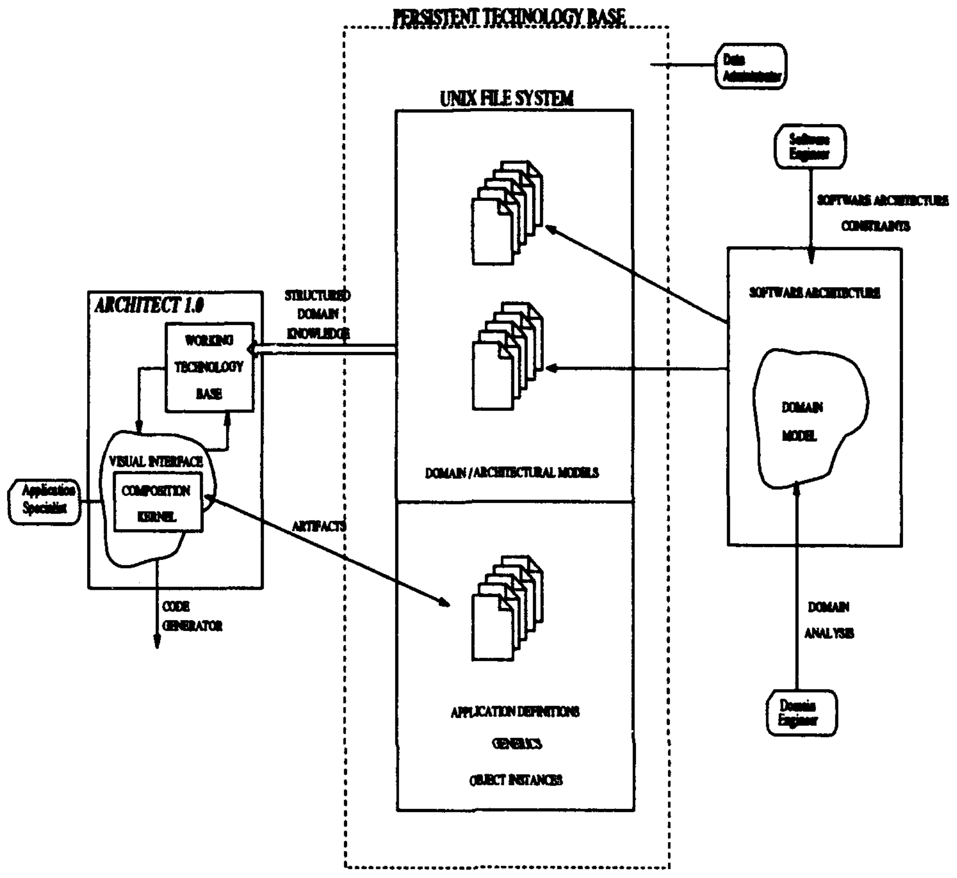


Figure 3.2 Architect 1.0 - Domain-Oriented Application Composition System

domains stored in the Persistent Technology Base get loaded into the Working Technology Base. Through the visual interface (35), the Application Specialist can manipulate domain primitives to compose and execute domain-oriented application specifications. The Application Specialist can also save newly composed applications to and restore saved applications from the Persistent Technology Base through the visual interface. The contents of the Working Technology Base are not persistent between Architect sessions, so any work the Application Specialist wants to keep must be saved using the visual system. In the remainder of this section we look in more detail at the key people and components shown in Figure 3.2 and introduced above.

3.2.1 Key People Interacting With Architect.

3.2.1.1 Domain Engineer.

A Domain Engineer possesses detailed knowledge about a domain and gathers all the information pertinent to solving problems in that domain (16:4). He models the real-world entities required to compose applications. A Domain Engineer also determines how, if possible, to model these entities within the architectural constraints specified by a Software Engineer (9).(1, 26)

3.2.1.2 Software Engineer.

A Software Engineer designs new software systems in an application domain (16:4). He is responsible for defining a formalized structure (software architecture) for the domain knowledge and providing the translation from the domain-specific terms to executable software (9). (1, 26)

3.2.1.3 Application Specialist.

An Application Specialist uses the domain and architectural models developed for a domain by the Software and Domain Engineers (16:4). He is a sophisticated "user" familiar with the overall domain. An Application Specialist understands what new applications must do to meet the requirements for a new system. He provides the application-specific information needed to specify and compose a new application. (1, 26)

3.2.1.4 Data Administrator.

A Data Administrator possesses detailed knowledge about the context within which the data in the technology base is used. He is responsible for ensuring the data stored in the technology base is current, accurate, and useful with respect to the needs of the users.

3.2.2 Key Components of Architect.

3.2.2.1 Software Architecture.

The software architecture Randour and Anderson chose to use for Architect was the Software Engineering Institute's (SEI) Object-Connection-Update (OCU) model (19). The basic premise of the OCU model is that real-world systems can be partitioned into a set of communicating subsystems. In the OCU model, subsystems consist of a controller, a set of primitive objects, an import area,

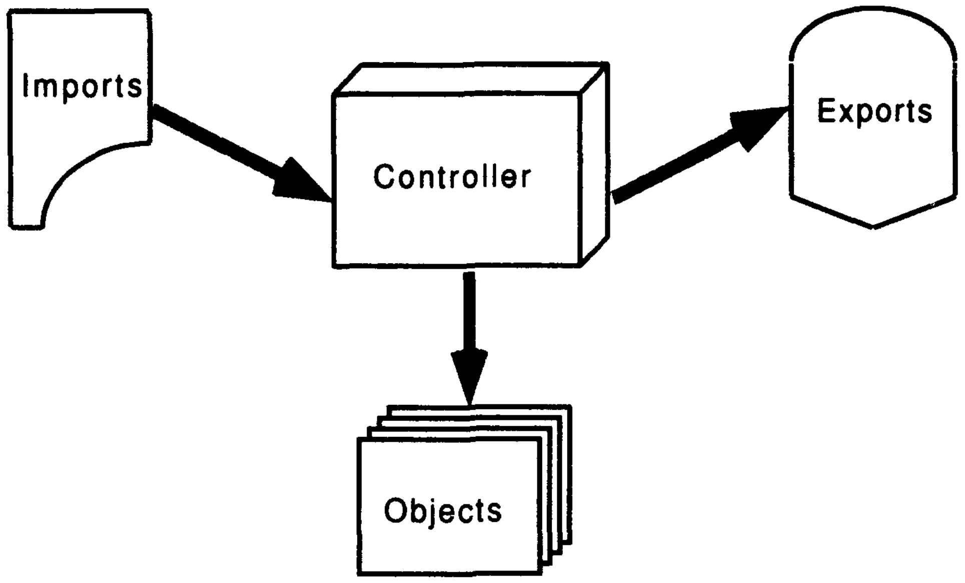


Figure 3.3 OCU Subsystem Construction

and an export area (19:17). Figure 3.3 shows a graphical representation of this relationship and is described below.

1. Controller – The Controller, as its name and position in Figure 3.3 both imply, is a critical element around which we build every subsystem. It encapsulates the mission of the subsystem and manages the connections between itself and a set of objects used to perform that mission.
2. Objects – An Object is an abstraction of a real-world entity, and maintains state to model behavior of that real-world entity.
3. Import Area – The Import Area is the focal point for the controller to get access to external data. The Import Area obtains this data from the Export Area of other subsystems.
4. Export Area – The Export Area is the focal point for making internal data available to other subsystems in an application.

Figure 3.4 shows the procedural interface for both controllers and objects, and we describe these interfaces below.

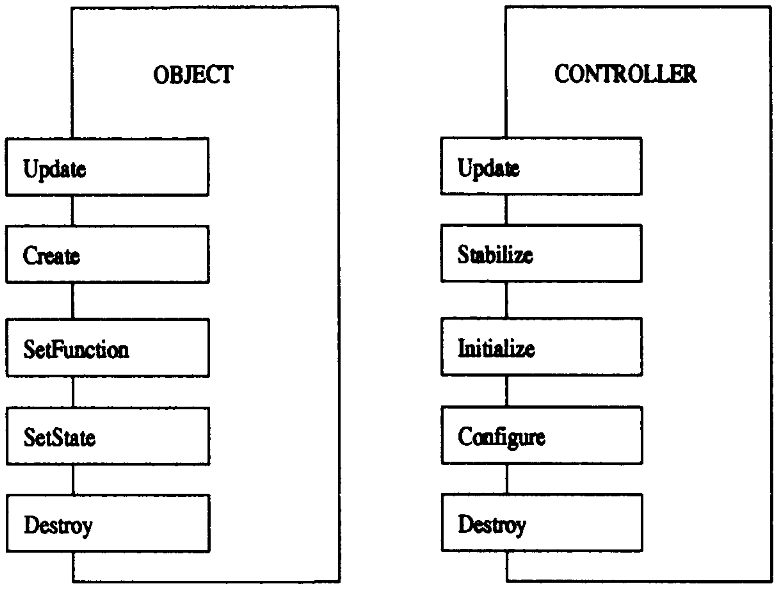


Figure 3.4 OCU Object and Controller Procedural Interfaces (19:18,20)

- Controller (19:19):

  1. Update – The controller obtains state data from the import area, calls the update functions of its lower-level primitive objects and subsystems in some predefined order, and provides new state data to its export area.
  2. Stabilize – Puts the system in a ready-to-operate state consistent with the current scenario.
  3. Initialize – Creates the objects and makes the connections for a subsystem.
  4. Configure – Sets up connections between the import and export areas and their corresponding input and output data.
  5. Destroy – Deallocates a subsystem's resources.

- Object (19:20):

  1. Update – Calculates the new state and outputs of the object based on input data and the current state of the object.
  2. Create – Allocates the resources necessary to create a new instance of an object.
  3. SetFunction – Modifies the way an object calculates its state data.

4. SetState – Bypasses the objects update function to set the state of an object directly.

5. Destroy – Deallocates the resources held by an object.

The types of data an object deals with are (19:20,21):

1. Input Data – Data used by an object to calculate its new state. Other subsystems or objects from within the same subsystem can import this data. The source of the data is transparent to the object.

2. Output Data – The externally visible state of an object made available by the most recent update. This state data can be exported to other subsystems or provided to other objects within the same subsystem. The destination of the output data is transparent to the object.

3. Attributes – Describe important characteristics of an object.

4. CurrentState – Data that, taken together, defines the current condition of an object. Not all of this data needs to be externally visible.

5. Coefficients – A special type of attribute used in update algorithms to calculate an object's state. Changes in coefficient values modify the behavior of an object's update procedure.

6. Constants – Also a special type of attribute. It cannot, however, be modified to change an object's behavior.

This brief discussion of the OCU software architectural model provides the groundwork for our discussion of design and implementation decisions, as well as our development of an object model for the OCU architecture in the next chapter. The SEI "Model-Based Software Development" article (19) is a good source of additional information on the "vanilla" OCU model, while several AFIT theses (1, 26, 35) provide more insight into the original AFIT implementation of the OCU model as the software architecture for Architect.

3.2.2.2 Domain. For an application domain to be used in an application composition system, the Domain and Software Engineers must first capture knowledge about the domain in an object model. Randour and Anderson used the Logic Circuits domain to validate Architect by composing and executing domain-oriented application specifications. We discuss the Logic Circuits domain further in Section 4.2, when we develop an object model for the domain using Rumbaugh's Object Modeling Technique (29). Randour and Anderson's theses (1, 26) also cover it in great detail.

3.2.2.3 Persistent Technology Base. The Persistent Technology Base of Architect is file-based. Artifacts created in the Working Technology Base are stored in separate files in a format that adheres to domain-specific and architecture grammars (26:4-7). This way, saved artifacts can be parsed back in to the Working Technology Base using the REFINE parse-file function. The capability to parse saved files back in to the Working Technology Base allows objects saved in the Persistent Technology Base to be reloaded. Randour describes the directory structure and naming conventions required by Architect in her thesis (26:Appendix A, section 5).

In addition to the application-specific artifacts composed by Application Specialists that are stored in the Persistent Technology Base, it also contains the OCU architecture model, all domain-specific models, and all the programs needed to make Architect execute. All of these files must be loaded into the Working Technology Base before the system can be used. Again, Architect software requires a pre-specified directory structure and naming conventions; these requirements are also found in Randour's thesis (26:Appendix A, section 5).

3.2.2.4 SOFTWARE REFINERY. Architect was implemented using SOFTWARE REFINERY™, a formal-based specification and programming environment. The SOFTWARE REFINERY environment includes a wide-spectrum programming language (REFINE) that supports set theory, logic, transformation rules, pattern matching, and procedures (27:1-2). Architect's Working Technology Base was implemented using SOFTWARE REFINERY's object base, while DIALECT, SOFTWARE REFINERY's language definition facility, was used to define Architect's Domain Specific and Architecture grammars. DIALECT is also used to parse compatible text files into the SOFTWARE REFINERY object base and to write object specifications out to file. Finally, INTERVISTA provides Architect's window-based graphical user interface. The SOFTWARE REFINERY environment was selected as the development environment for Architect because its powerful integrated tool set supports rapid prototyping so well (26:3-23).

3.2.3 Functionality of Architect. This section describes the processes of capturing new domain knowledge and composing systems under Architect, using the visual interface developed by Weide (35).

3.2.3.1 Capturing New Domains. To capture new domains, the Domain Enginee· and Software Engineer need to develop SOFTWARE REFINERY source code files. A separate file is created for each primitive in the domain, for the overall object model, and for a domain-specific grammar. Also, the Software Engineer needs to create a file describing the visual information for the domain primitives. This file is written in a visual specification language that can be parsed in using a visual description grammar (35). Once these files are created, compiled, and verified, they are placed in the technology base directory structure.

3.2.3.2 Composing Systems. To start an Architect session, the Application Specialist starts a REFINE session in the root of the directory structure referred to in section 3.2.2.3. By loading a startup file at the lisp prompt, the entire Architect environment is loaded. This includes loading DIALECT, INTERVISTA, and all the executable Architect code, as well as the OCU architecture model and all domain-specific models, grammars, and visual descriptions. The user is then presented with a window-based menu system, and the session begins. At this point, the Application Specialist can load in previously saved application compositions, or begin new ones. Once a session contains an application composition that has been created and/or modified, the Application Specialist can save it to the technology base through a menu option.

3.3 Requirements Analysis Results

This section provides the results of our requirements analysis for improving Architect by moving towards a domain-oriented application composition environment similar to the one shown in Figure 3.1. Our implementation addresses the following list of identified requirements:

1. Identify Artifacts:

   The purpose here is to establish what the data requirements are for the technology base. We began by analyzing the current implementation of Architect to identify implementation-specific requirements, as well as requirements for domain-oriented application composition systems in general.

2. Select OODBMS for Technology Base Implementation:

   On the basis of the data requirements we identified, and our analysis of how the data will be used, we needed to select the best of the available OODBMSs for our implementation. We needed to determine which of the OODBMS capabilities identified in our literature search are important for this application area, and evaluate each of the three systems available to us based on these criteria.

3. Investigate Database Communications Interface:

   We needed to identify and evaluate options for the communications backplane that would allow other tools in the new environment to communicate with the OODBMS technology base.

4. Provide a Central Repository for Software Artifacts:

   One of the major goals of this research was to replace the file-based technology base of Architect with an object-oriented database implementation. This helps to bridge the gap between the current capabilities of Architect, and some of the other J-MASS requirements such as the CONFIGURE SIMULATION and EXECUTE SIMULATION functionality.

   (a) Develop Object Model for Software Architecture:

       Because we are building up an environment that may have more than one composition system implementation, possibly based on the same software architecture, we required an implementation-independent object model for software architectures. The goal here was to produce a database schema definition of the OCU model that is not Architect dependent.

   (b) Develop Object Model for Application Domains:

   Since application domains are also artifacts that may be reused by multiple composition systems, possibly based on different architectures, we had to develop a schema that describes the general domain knowledge. This general domain schema can then be extended for combinations of software architectures and specific implementations using the multiple inheritance feature of OODBMSs.

   (c) Add Database Storage and Retrieval Functionality to Architect:

   The current functionality of Architect must be extended to communicate with the database for storage and retrieval of composed artifacts. This should be done in a incremental development process to build up and verify these capabilities.

   (d) Provide for Storage of Miscellaneous Software Artifacts:

   Besides storing artifacts produced by Architect, we found we could also store artifacts used for developing Architect, and other composition systems. We identified what these artifacts were (source code, compiled code, visual interface objects, etc.), and provided the schema and methods for their storage and retrieval.

5. Isolate Domain Definition from Implementation:

   We found during our requirements analysis that a database implementation put new responsibilities on the Domain Engineer. Now, not only did the domain need to be defined in the object base of the application as before, but the same information must be communicated to a database administrator to define the database schema for storing artifacts of the domain. If we could capture this domain knowledge in the database in the form of a model description, perhaps we could automate some of these functions for the Domain Engineer.

   (a) Develop Meta-Model for Capturing Domain Knowledge:

   Develop a meta-model to formally capture domain model definitions that could be used by automated transformation processes.

(b) Provide Capability to Populate Database With Domain Definitions:

Provide the Domain Engineer with the capability to input and modify domain definitions after completing his domain analysis. This can be a prototype used in later research for the Technology Base Extender shown in the upper-left corner of Figure 3.1.

(c) Automate Transformations of Domain Model to Database Schema:

Using rapid prototyping, incrementally develop an automated process to transform the abstract domain definition into a database schema for specific software architectures.

(d) Automate Transformations of Domain Model to Object Base Definition:

Using rapid prototyping, incrementally develop an automated process for transforming the abstract domain definition into source code necessary to describe the domain in a specific application object-base environment.

6. Support Concurrent Research Efforts:

As the Architect system evolves with the results of other research at AFIT, make sure that our extensions support the new changes. Also, maintain close coordination with researchers to identify any new database capabilities or benefits that would enhance their research.

3.4 OODBMS Comparison

This section discusses our selection of an OODBMS. There were three OODBMSs available to us: MATISSE, OBJECTSTORE, and ITASCA. The designers of all three systems designed their products from the ground up to store objects, and to support multiple inheritance. After highlighting the features, capabilities, and differences of the three systems, we discuss the choice we made, and the reasons for making the choice.

3.4.1 MATISSE. The MATISSE OODBMS operates under the client-server architecture. The MATISSE server is a general purpose object manager, operating as a multi-threaded back-end server, that manages a repository of persistent objects (the database), and communicates with clients through a remote-procedure-call (RPC) interface (15). MATISSE supports multiple inheritance, versioning, on-line dynamic schema evolution, and ensured referential integration (13). MATISSE, using intrinsic versioning, makes a new version of an object with every change.

MATISSE, developed using C++, easily interfaces to C or C++. The MATISSE semantic data model is language independent; the application programmer interface (API) uses the C language. The API is strictly a functional interface, with no extensions to the C language; as such, any programming language that supports external calls can interface with MATISSE. Since schema objects are created in the database through a functional interface, they can be modified or deleted at run-time, allowing a great deal of on-line schema modification.

MATISSE is an active database, supporting the execution of methods within the database environment. The database, however, does not store the methods. The methods are part of the application program. This means that changes to the methods require re-compilation of the C source code.

MATISSE support tools include a query language, a 4GL capability, a schema editor, and an object editor. Developers and end-users can use the object editor to view any object in the database using a standard form or an object window (14).

3.4.2 OBJECTSTORE. The OBJECTSTORE OODBMS also operates under the client-server architecture. It is an example of an OODBMS built by extending an object-oriented programming language to include database functionality. Written in C/C++, its data manipulation and management language (DMML) is an extended C++ environment that includes both C and C++ library interfaces (13).

Extensions to the language allow the programmer to treat persistent data and transient data the same. Persistence is not part of the type of an object (i.e., there is no need to inherit from a special "persistent object" base class), therefore different instances of the same class may be persistent or transient within the same program (18). The language extensions require a pre-compiler that converts extended language statements to standard C statements.

ObjectStore provides limited support for composite objects and schema evolution. Since the C source code contains the schema definitions, schema modifications require re-compilation. There is a customized query language but no 4GL capability.

ObjectStore is a static database; any request results in the database assembling the object and then passing it to the application. The most common use of ObjectStore is in providing persistent data to C/C++ applications.

3.4.3 Itasca. Like the other two OODBMSs, Itasca runs under the client-server computing architecture. Although written with a combination of C and Lisp, the Itasca DMML is essentially an interpreted Lisp environment (13). Like Matisse, Itasca is an active database allowing methods to be executed within the database. In addition, Itasca can store the methods in the database with the objects.

Language interfaces for Itasca include C, C++, Lisp, Common Lisp Object System (CLOS), and, under development, Ada (13). Programming languages can easily use objects created by any other language (13). The database stores methods with the schema, in interpreted Lisp form; thus, method modifications do not require re-compilation. Itasca also supports extensive schema modification without shutting the database down.

Itasca supports both shared and private databases in a fully distributed database environment with complete location transparency. Extensive object versioning, passive and active change notification, and elaborate locking on the object model are a few of Itasca's strengths.

Itasca comes with three OSF/Motif based productivity tools: a database administration tool, a dynamic schema editor, and an active data editor. The database administration tool provides capabilities for site maintenance including backups, addition and modification of user security, and system performance monitoring. The dynamic schema editor provides a simple mechanism for modifying class definitions, to include attributes and methods. The active data editor allows views and modifications of objects in the database, and provides a desktop work area for query storage.

3.4.4 Selection of an OODBMS. We chose the ITASCA OODBMS for our implementation. We felt the dynamic schema evolution feature without re-compilation of methods best met our need for a rapid prototyping capability. The application we were extending with database functionality runs under SOFTWARE REFINERY, a product that runs in the Allegro Common Lisp environment. Since ITASCA also runs in a Common Lisp environment, we did not have to translate to a target language. The storage of methods in the database and support for a number of languages accessing the same objects will support reuse, an important requirement in an evolving prototype system.

ITASCA provided a remote Lisp interface for its Lisp API that supported both Allegro Common Lisp and Lucid Common Lisp. The remote Lisp interface, as delivered, supported Allegro versions after 4.0. Since AFIT's current version of SOFTWARE REFINERY runs on Allegro version 3.1.13, we had to make a few modifications to the source code to make it backward compatible. The modifications were syntactic in nature, and were provided by Allegro technical support personnel. SOFTWARE REFINERY has scheduled a new release of REFINE that will operate under version 4.1 of Allegro Common Lisp. This may make the use of ITASCA even more valuable, as the CLOS interface may allow persistence of objects in the database, making persistence transparent to application programs.

3.5 Conclusion

We began this chapter by analyzing and explaining how the current implementation of Architect operates in a stand-alone mode. This provides the reader with a good reference point for the state of the system before our addition of an OODBMS technology base. Then, based on the results of our literature search and analysis of Architect, we identified the requirements and goals we developed for creating a domain-oriented application composition environment in which Architect could work. A description of this new environment is provided later in this document when we present our results and conclusions. We compared MATISSE, OBJECTSTORE, and ITASCA OODBMSs, and discussed our rationale for using ITASCA for our technology base. Having identified our requirements, we begin our design and implementation.

IV. Design and Implementation of an OODBMS Technology Base

4.1 Introduction

This chapter describes the steps we took to design and implement an OODBMS technology base for Architect. Section 4.2 discusses our selection of a validating domain with which to test our system. In that section, we identify the types of artifacts we want to store in the technology base, develop an object model for the domain, and develop storage and retrieval tests to execute against the finished product. Section 4.3 discusses our investigation and selection of a communications method between the database and application programs. Section 4.4 covers the development of an object model for the software architecture and its subsequent conversion to a database schema. Section 4.5 covers the process we went through to incorporate the OODBMS version of the technology base into AVSI. Finally, we discuss the development of the meta-model used by a Domain Engineer to generate the domain-specific portion of the database schema and the formal domain definition for the application composer.

4.2 Selection of a Validating Domain

One of the decisions we had to make was what application domain to use to test our OODBMS implementation of the technology base. One factor we considered in that decision was our desire to transition from the file-based technology base to an OODBMS implementation without losing domain analysis work done by previous researchers. Transforming an existing technology base of software artifacts had the added benefit of providing us with the test data against which to check our new implementation. Appendix A contains a description of the technology base of primitive objects for the logic circuits domain. Appendix C contains applications built from the primitives. These software artifacts were part of the baseline system we inherited at the start of our research. Since other researchers were already extending the technology base to other domains, we saw little benefit in developing a new domain with which to test our implementation of the technology base. For these reasons, we decided to use the existing logic circuits domain as our validating domain.

The domain itself was not the focus of this research. We refer the reader interested in the logic circuits domain, and the analysis of that domain, to (1) and (22); these two sources provided us with enough knowledge of the domain to conduct our research.

4.2.1 Identification of Artifacts in Selected Domain. The software artifacts residing in the baseline technology base were of two types: primitives and applications.

- PRIMITIVES
  - AND-GATE
  - OR-GATE
  - NAND-GATE
  - NOR-GATE
  - NOT-GATE
  - SWITCH
  - LED
  - COUNTER
  - DECODER
  - 4-INPUT MULTIPLEXER (MUX)
  - JK-FLOP-FLOP
  - HALF-ADDER

  The following is a brief description of the composed applications (refer to Appendix C for more information).

- APPLICATIONS
  - TEST
    This is a simple test application that uses 2 switches, 2 "and" gates, and 2 LEDs.
  - ADDER
    This application is a full adder. It is built using 2 subsystems that are half adders, built out of "and", "or", and "not" gates.
  - DECODER
    This application provides a test of a "decoder" primitive by providing 3 input switches and 8 output LEDs.
  - ADD-AND-DECODE1
    This application tests the combination of an adder and a decoder. The adder is a subsystem composed of 2 half-adder subsystems, each composed of "and" gates, "or" gates, and "not" gates. The decoder is also a subsystem, composed of "and" gates and "not" gates.

- ADD-AND-DECODE2

  This application is similar to the ADD-AND-DECODE1 application mentioned above, except the decoder subsystem is implemented as a decoder with output (i.e., the LEDs have been moved down under the control of the decoder subsystem).
- BCD-ADDER

  This application tests a subsystem implementation of a BCD-adder using primitive gates and half-adders.
- BINARY-ARRAY-MULTIPLIER1

  This application implements a binary array multiplier as a top level subsystem, composed of "and" gates, "not" gates, switches, leds, and 2 half-adder subsystems, each of which is composed of "and" gates, "or" gates, and "not" gates.
- BINARY-ARRAY-MULTIPLIER2

  This application implements a binary array multiplier similar to BINARY-ARRAY-MULTIPLIER1 listed above, except that primitive half-adders are used instead of lower-level subsystems composed of primitive gates.
- THREE-TO-EIGHT-DECODER

  This application composes "and" gates and "not" gates into a subsystem implementation of a 3-to-8 decoder.

4.2.2 Development of Object Model for Selected Domain. Because the object model of a domain plays such a critical role in our development of a database schema and the object hierarchy in the object base, our next step was to use Rumbaugh's object modeling technique (29) to produce an object model of the logic circuits domain. Figure 4.1 is the object model we developed by analyzing the artifacts in the baseline technology base and referring to our two primary sources of domain knowledge (1, 22) discussed earlier.

The Architecture Primitive at the top of Figure 4.1 is not truly a part of the logic circuits domain, but is actually a class found in the software architecture model of the application-composition system. This association points out that the logic circuits domain model inherits from the architecture object model to produce a complete object model of a domain that conforms to the structure of a specific architecture. Please note too, the legend in the lower left hand corner of Figure 4.1. A clear box represents a concrete class and a shaded box represents an abstract class. An abstract class is never instantiated; it merely provides a high-level definition that one or more subclasses can expand upon. A concrete class inherits attributes and operations from parent abstract classes, and can be

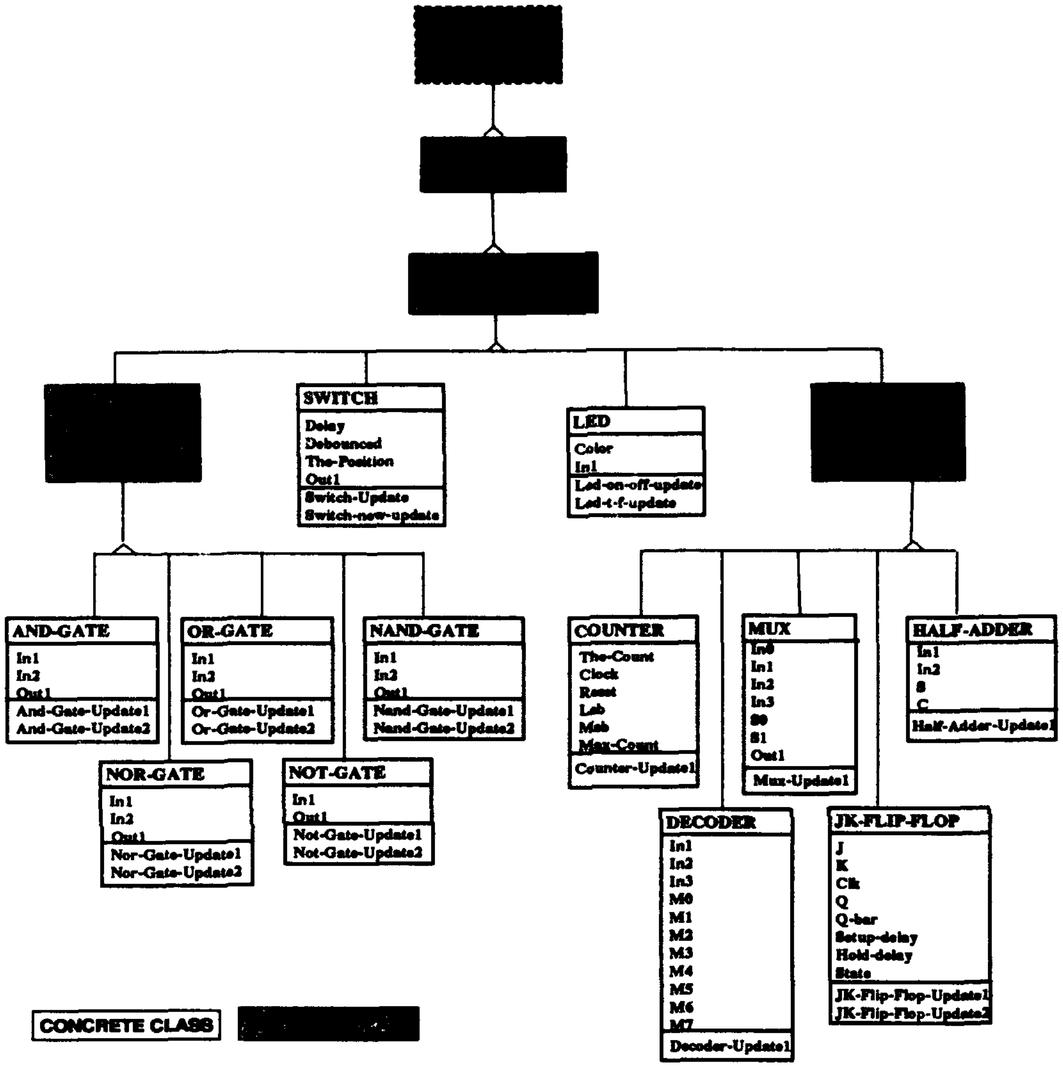


Figure 4.1 Object Model of Logic Circuits Domain

instantiated to create objects of that class. As an example, it makes no sense in the logic circuits domain to create a GATE object unless you further defined it as an AND-GATE, OR-GATE, NAND-GATE, NOR-GATE, or NOT-GATE object. On the other hand, when an AND-GATE object is instantiated, it inherits attributes Delay, Mil-Spec?, and Power-level from abstract class GATE, as well as the attribute Manufacturer from abstract class CIRCUIT-ARTIFACT.

Discussing Figure 4.1 from top to bottom and left to right, the first class we come across in the logic circuits domain (discounting Architecture Primitive) is the abstract class labeled Circuits. This is a container class we require in our implementation to aid in extracting a list of available domains from a technology base full of domains. We require that it have no attributes or operations associated with it. Its sole purpose is to provide a handle by which to access a domain. In Section 4.6 we explain the reasons for this implementation decision when we discuss the object model for defining domains.

The next object class in Figure 4.1 is the CIRCUIT-ARTIFACT class. CIRCUIT-ARTIFACT has one attribute, Manufacturer, which all other object classes in the object model inherit. We feel it is important to note at this point that this is how we chose to model this domain, and that other solutions can provide the same results. Figure 4.2 for example, is the same object model with the abstract class CIRCUIT-ARTIFACT removed. To propagate the Manufacturer attribute to all object classes in the domain, we had to replicate it in the GATE, SWITCH, LED, and COMPONENT classes. We chose to take full advantage of inheritance and add the CIRCUIT-ARTIFACT class with the Manufacturer attribute. Referring again to Figure 4.1, we see that a CIRCUIT-ARTIFACT is further defined as a GATE, SWITCH, LED, or COMPONENT. An AND-GATE, OR-GATE, NAND-GATE, NOR-GATE, or NOT-GATE is a more detailed definition of a GATE, while a COUNTER, MUX, HALF-ADDER, DECODER, or JK-FLIP-FLOP is refined from a COMPONENT. There may be many good alternatives to our object model of the logic circuits domain, but the model in Figure 4.1 is the one we used in our research.

4.3 Database Communications Platform

Our goal was to investigate current implementations of CORBA-compliant database interfaces. Unfortunately, we found that the Object Management Group had put a lot of effort into developing a standard interface, but as yet there are only a few commercial applications addressing the standard. The implementation of a CORBA-compliant interface is composed of a number of pieces (23), an object request broker (ORB) for the database and an interface definition language (IDL) being two of the key items.

Although we did not find a CORBA-compliant interface to use, we investigated two possibilities to build a communications interface on.

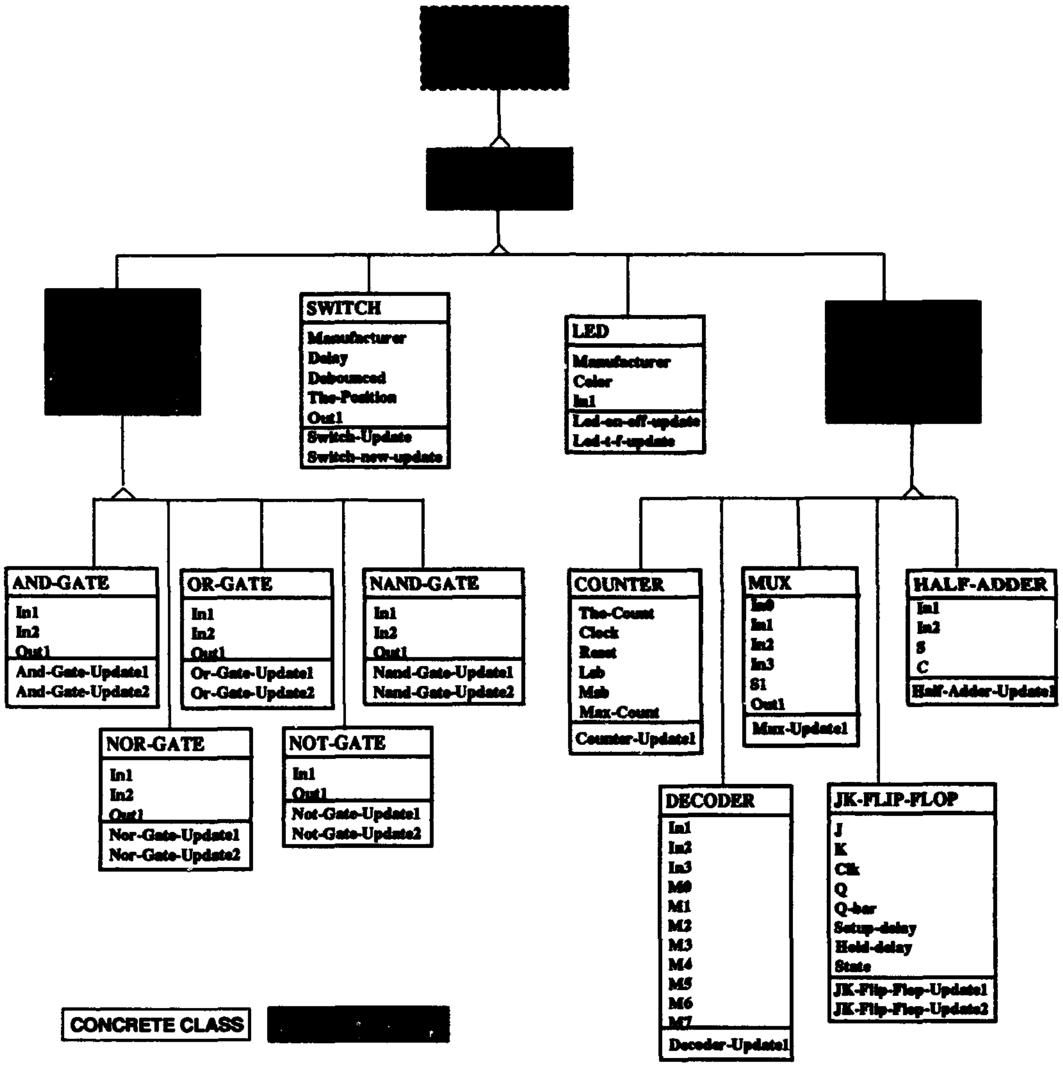


Figure 4.2 Alternate Object Model of Logic Circuits Domain

4.3.1 ToolTalk. The first product we investigated was the TOOLTALK™ service, from SUNSOFT™, an inter-application message system designed to promote cooperation among independently developed applications across networks. The TOOLTALK service can be used in two ways (32):

- Applications use the TOOLTALK service directly, calling functions contained in the TOOLTALK API library to create, send, and receive messages.

- Applications could use a "service" built on top of the TOOLTALK service. These application services would then use the TOOLTALK service as a communications backplane.

Currently the TOOLTALK service messages are file-based, although SUNSOFT is incorporating it in their Project DOE, a move towards a distributed object paradigm. Services are accessed by C library routines. In order to use TOOLTALK for our research, an message interface would have to be developed, in C, for the application as well as the database.

As SUNSOFT's Project DOE matures, the distributed object paradigm would possibly make TOOLTALK a good development tool for a CORBA-compliant communications backplane. However, the development of a CORBA-compliant interface was beyond the scope of this research effort.

4.3.2 ITASCA's Remote Lisp API. The release of ITASCA at AFIT contained a user support library where we found, among other things, a remote Lisp application programmer interface (API). This API supports the client/server model of communications over a network, allowing multiple application programs running on various network workstations to communicate with the ITASCA database server running on a single workstation. The API supported Lisp interfaces to either Lucid Common Lisp (the Lisp that ITASCA runs under) or Allegro Common Lisp. The fact that SOFTWARE REFINERY runs under Allegro Lisp version 3.1.13 made this an attractive possibility for our implementation. Two of the key features were that communications were already supported, and that a common language was supported by ITASCA and SOFTWARE REFINERY.

Using our rapid prototyping approach, we began investigating and familiarizing ourselves with ITASCA using the Allegro Common Lisp software (version 4.1) available on our network and the remote Lisp interface software.

Our next phase was to test the remote Lisp interface within the EMACS environment that SOFTWARE REFINERY runs under (Allegro version 3.1.13). We found that the remote API had been written to support Allegro versions 4.0 and greater. Specifically, the interface to stream sockets had changed after version 4.0 was released. We contacted both ITASCA and Allegro customer support centers, and received the necessary changes to make the software operate under version 3.1.13 of Allegro Common Lisp.

Our last phase was to integrate the remote Lisp API directly within the REFINE environment. We found that we could easily transfer data back and forth between a REFINE application program and an ITASCA server running on another network workstation.

4.3.3 Conclusion. Based on the success of the ITASCA remote Lisp interface in dealing with REFINE application programs, we selected this option to represent our communications backplane for this first prototype version of a domain-oriented application composition environment.

4.4 Target Architecture Description

One of the major goals in our research was to give Architect the ability to store and retrieve artifacts using the database. Since Architect already had an object model developed for the OCU architecture, we could have simply replicated the same object model in our database schema. After a close analysis of the AVSI system, we found its developers designed the object model with ease of population and execution in mind. The flattened tree structure of AVSI, with applications, subsystems, and primitive objects at the same level, allows DIALECT to easily parse artifacts in from and out to text files. There is, however, some loss of the hierarchy inherent to the OCU model. AVSI's method of referencing object relationships through unique object names within each application makes execution of applications easier; however, a database storing multiple applications cannot maintain this unique name requirement. Subsystems and primitives from several applications could have the same names, and the database requires names be unique throughout.

4.4.1 Object-Connection-Update (OCU) Architecture. Besides storing artifacts for reuse by AVSI, another goal was to allow access to the same data by future applications. With this in mind, we decided to design our own object model for OCU that was less

Software Refinery implementation specific, and to use this model for building our database schema.

Following Rumbaugh's (29) object modeling technique (OMT), we identified the key objects of the OCU model, and the associations between them. The basic hierarchy of the OCU model is an application, controlling a number of top-level subsystems, each of which controls any number of primitive objects and lower level subsystems. To capture this recursive nesting of subsystems, we abstracted primitive objects and subsystems under a new superclass called an element. Each subsystem controls any number of elements, any of which can be another subsystem which in turn controls any number of elements, and so on.

4.4.2 OCU Object Model Development. In the model we developed (Figure 4.3), associations between objects are maintained through the object identifiers generated by the system, as opposed to the value of a name attribute. This eliminates the need for uniquely named objects, and reduces any overhead associated with changing the value of an object's name.

In the Architect system, all the inputs and outputs of the primitive objects controlled by a subsystem are imports and exports of the subsystem. We felt this to be an implementation decision that the database should not carry over, allowing future applications greater flexibility. In our interpretation of the OCU model, an import of a subsystem is an externally available input port for the subsystem, which internally provides its value to one or more inputs of primitives, and/or imports of lower level subsystems, that the subsystem controls. An export of a subsystem is an externally available output port with an internally controlled value.

This object model is not complete. The definition of what exactly an application is, and how it executes, is the subject of other research currently under study at AFIT (33, 36). In our current version, an application simply consists of a top-level subsystem that executes in a non-event driven sequential manner. Welgan (36) and Waggoner's (33) research expands application definitions to include event-driven sequential and time-driven

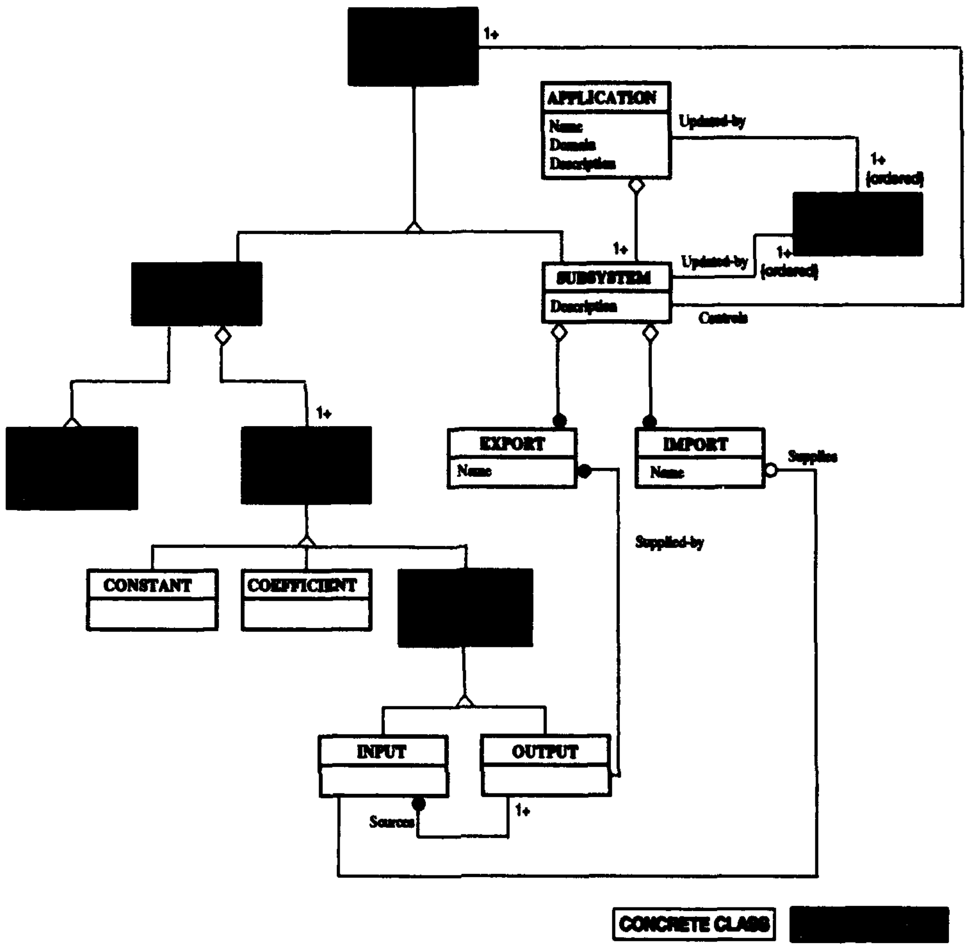


Figure 4.3 Object Model of Object-Connection-Update (OCU) Architecture

sequential operating modes. The dynamic schema modification characteristic of ITASCA will allow the addition of future research results with little or no impact.

Our representation of the OCU architecture model is shown in Figure 4.3. Elements of application domains are defined as subclasses of our PRIMITIVE class. In this way, the two domain models become one as illustrated in Figure 4.4.

4.5 Incorporation of ITASCA *in AVSI*

With all the pieces defined, the last step was to incorporate the database functionality into the AVSI system. As far as the Application Specialist is concerned, the database

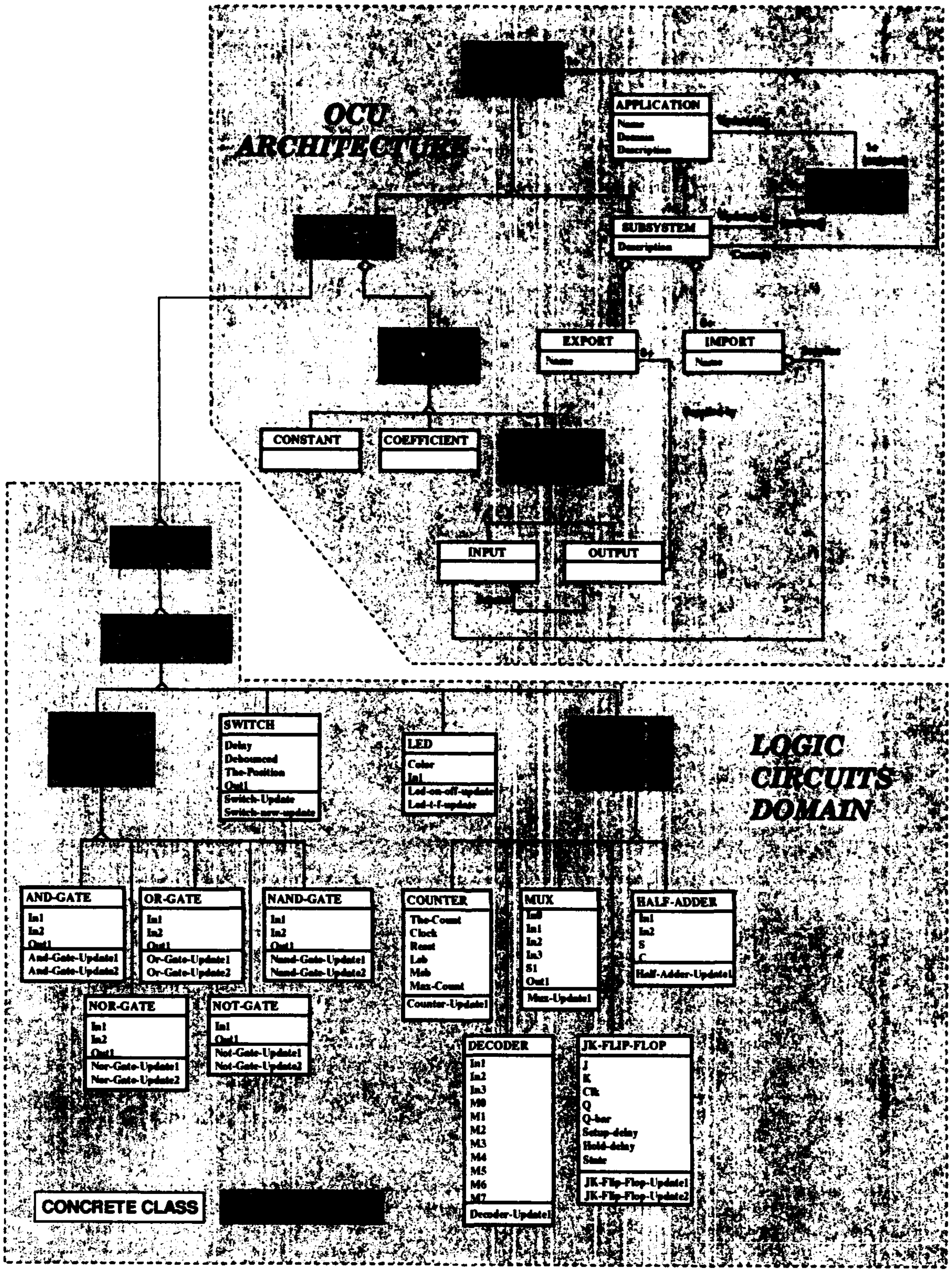


Figure 4.4 Combined Object Models Showing Circuits Inheriting from OCU

functions the same as the file-based technology base did. Our changes are virtually transparent. There is no loss of current AVSI capability, but new options are available for saving and retrieving composed applications in ITASCA.

A simple example of a composed application is a light, consisting of a switch and a LED. This application, which is used as an example in Appendix E, will be referred to throughout this section.

4.5.1 Background. The methodology for saving and restoring composed applications relied heavily on the tree structures in the object models of both AVSI and ITASCA. Although there is not a one-to-one correspondence between the two structures, they are similar in the information they contain. The problem, then, was to figure out the mapping from one to the other, and vice versa.

The root of the tree structure of an AVSI composed application is an object of type SPEC-OBJ (Figure 4.5). The AVSI software enforces the structure of this tree through its semantic checks. Our assumption was that any application the user wants to save to the database has already passed the AVSI semantic checks. The second level of the tree structure contains one, and only one, object of type APPLICATION-OBJ. A DESCRIPTOR-OBJ for the application, a SUBSYSTEM-OBJ for each OCU subsystem in the application, and one object that is a subclass of PRIMITIVE-OBJ for each primitive object in the application are also on the second level. The third level of the tree extends down from the SUBSYSTEM-OBJ objects and contains objects of type IMPORT-OBJ and EXPORT-OBJ associated with each subsystem; one for each input and output, respectively, of the primitive objects the subsystem controls. Additionally, the sets of objects that are subclasses of class STATEMENT-OBJ form the "Update" functions for the APPLICATION-OBJ and all SUBSYSTEM-OBJs. The fourth, and final, level of the tree extends down from the IMPORT-OBJ and EXPORT-OBJ objects and contains objects of type SOURCE-OBJ and TARGET-OBJ.

As noted before, the architectural structure of AVSI is not contained in the tree itself, but in unique name references in the objects themselves. The APPLICATION-OBJ contains a list of the highest-level object(s) of type SUBSYSTEM-OBJ in the application in

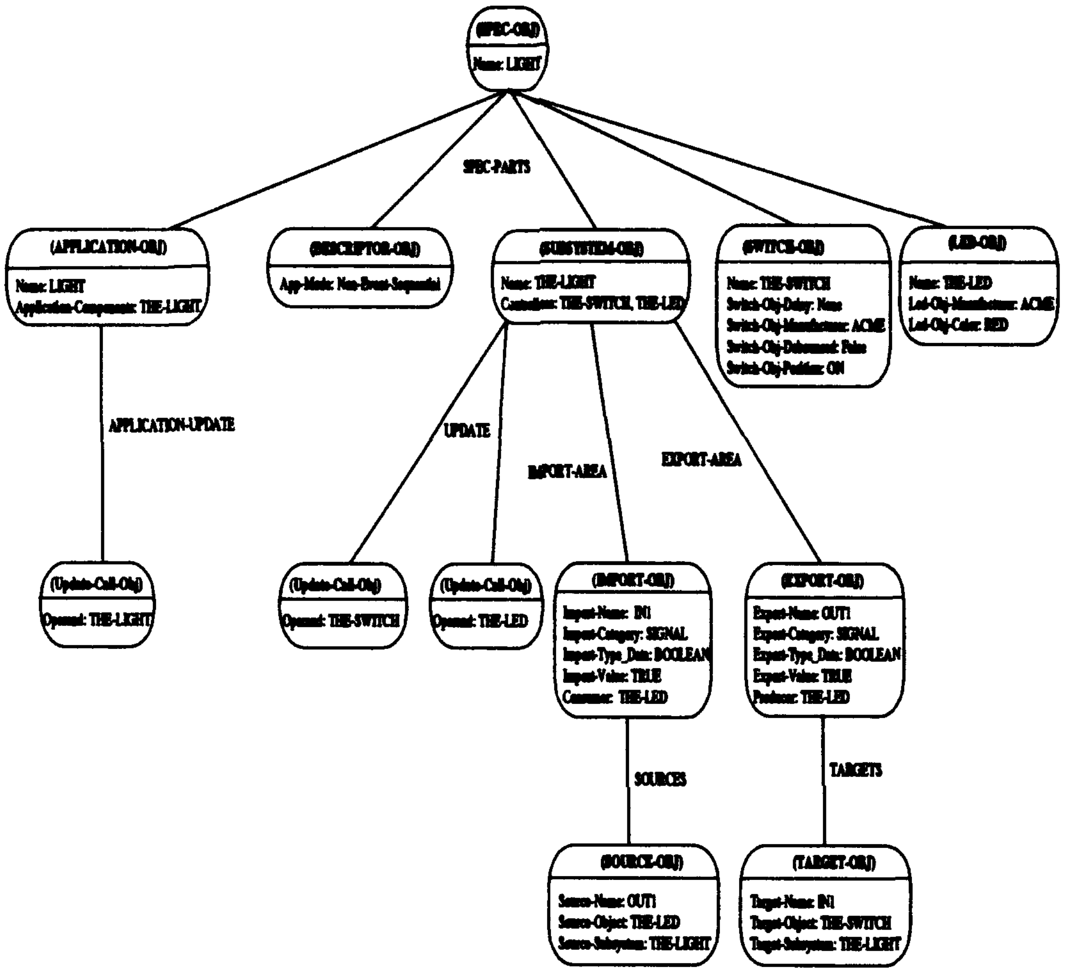


Figure 4.5 AVSI Instance Diagram for Light Application

its APPLICATION-COMPONENTS attribute. For our "Light" example, this consists of the one subsystem named "THE-LIGHT" (Figure 4.5). Each SUBSYSTEM-OBJ contains a list of names of the PRIMITIVE-OBJ objects and lower-level SUBSYSTEM-OBJ objects it controls in its CONTROLLEES attribute. For our example, this is the names "THE-SWITCH" and "THE-LED" (Figure 4.5). Each EXPORT-OBJ keeps the name of the PRIMITIVE-OBJ it belongs to in its PRODUCER attribute, and each IMPORT-OBJ keeps the name of the PRIMITIVE-OBJ it belongs to in its CONSUMER attribute. Each SOURCE-OBJ is associated with an EXPORT-OBJ somewhere within the application and contains the following three things.

1. The name of the EXPORT-OBJ in its SOURCE-NAME attribute,
2. The name of the PRIMITIVE-OBJ that produces the export value in its SOURCE-OBJECT attribute, and
3. The name of the SUBSYSTEM-OBJ that controls the primitive object in its SOURCE-SUBSYSTEM attribute.

The TARGET-OBJ is similarly associated with an IMPORT-OBJ somewhere within the application and contains the following three things.

1. The name of the IMPORT-OBJ in its TARGET-NAME attribute,
2. The name of the PRIMITIVE-OBJ that consumes the import value in its TARGET-OBJECT attribute, and
3. The name of the SUBSYSTEM-OBJ that controls the primitive object in its TARGET-SUBSYSTEM attribute(Figure 4.5).

Because ITASCA uses a single name space for class names, we prepended the string "OCU-" to class names derived from Figure 4.3. This eliminates future collisions when other architectures might be explored. The root of the tree structure for a saved application in ITASCA is an OCU-APPLICATION object (Figure 4.6). Its leaves are the set of top-level OCU-SUBSYSTEM objects contained in the application. Each OCU-SUBSYSTEM object contains leaves of lower-level SUBSYSTEM-OBJ objects and instances of OCU-PRIMITIVE subclasses (inherited in the domain-specific object model) that it controls.

In turn, each instance of an OCU-PRIMITIVE subclass contains leaves for its inputs and outputs, contained in instances of OCU-INPUT and OCU-OUTPUT respectively. Contained in each OCU-INPUT object is a referential attribute that identifies its source OCU-OUTPUT(s). Note, also, that all references between objects in this structure are based on the ITASCA-generated object identifiers, and do not rely on the names of the objects themselves.

Each element in the OCU model (applications, subsystems, and primitive objects) has an associated update function. The Domain Engineer defines the update functions for primitive objects, and these do not change during a session of AVSI; they become a part of

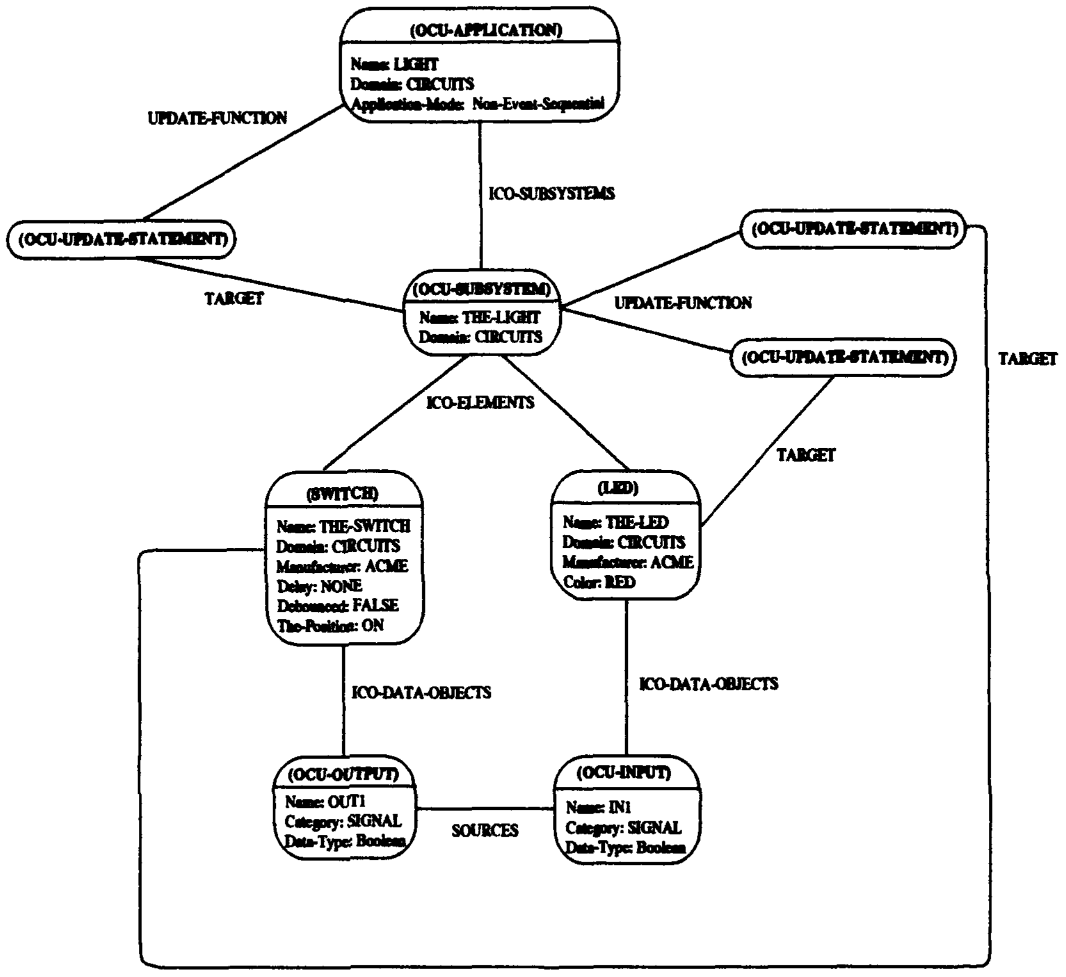


Figure 4.6 ITASCA Instance Diagram for Light Application

the domain-specific object model stored in the technology base. When saving applications to the database, primitive objects store only the name of their update function. The other update functions (applications and subsystems) are composed within the AVSI session, and hence need to be saved. We store these functions as an ordered sequence of OCU-STATEMENT objects in the UPDATE-FUNCTION attribute of OCU-APPLICATION and OCU-SUBSYSTEM objects (Figure 4.6).

4.5.2 Saving Composed Applications into ITASCA. To save an application composed with AVSI to the database, we first walk the AVSI tree structure in a top down fashion. We create a corresponding OCU-APPLICATION object in the database for the APPLICATION-OBJ, and keep its object identifier as a reference to the root of the database tree. In the same fashion, we create a corresponding OCU-SUBSYSTEM object for each SUBSYSTEM-OBJ, and an object of the same class name for each instance of a PRIMITIVE-OBJ subclass object. At this time, we copy all the attributes the Domain Engineer defined to be "edit-attributes" of the domain-specific primitives from the REFINE (AVSI) object base to the ITASCA objects. As we create all the database objects up to this point, we save the object-identifier created by ITASCA in a mapping whose domain is the name of the corresponding AVSI uniquely named object.

At this point, we are ready to add the structure to our database application. Starting with the APPLICATION-COMPONENTS attribute of the OCU-APPLICATION object, we place the object identifiers these names map to in the ICO-SUBSYSTEMS attribute of the OCU-APPLICATION. We also make a call to a subroutine that saves the APPLICATION-OBJ's update function. Next, we iterate over all the SUBSYSTEM-OBJ objects contained in the tree, calling a subroutine to save the SUBSYSTEM-OBJ's update function. Another subsystem call iterates over the set of all SUBSYSTEM-OBJs, using the CONTROLLEES attribute names to store the corresponding object identifiers in the ICO-ELEMENTS attributes of each OCU-SUBSYSTEM object.

The final step to complete our hierarchy is to iterate over all the IMPORT-OBJ and EXPORT-OBJ objects, creating OCU-INPUT and OCU-OUTPUT objects in the database and linking them to the correct OCU-PRIMITIVE objects based on the name attributes stored in the AVSI objects. We create all the OCU-OUTPUT objects first, so when we create the OCU-INPUT objects we can associate them with the correct source OCU-OUTPUT.

Note that we consider all the work done to save an entire application to be one transaction for the database. If at any point in the process something goes wrong, we abort the entire transaction, returning the database to a non-corrupted state. If the entire process succeeds, we commit the transaction, and the database contains the saved application.

4.5.3 Loading Composed Applications from ITASCA. Since we save the entire OCU hierarchy within the database, we can load the application back into the REFINE object base in a single, top-down pass through the tree. After selecting an OCU-APPLICATION from the database, we create a SPEC-OBJ in the REFINE object base to be the tree root, and create an APPLICATION-OBJ and place it as a leaf. We iterate over the ICO-SUBSYSTEMS attribute of the OCU-APPLICATION, making calls to a subprogram for creating the appropriate SUBSYSTEM-OBJ objects. This subprogram also creates the instances of PRIMITIVE-OBJ subclasses, as well as calling itself recursively to build lower level subsystems. Since all the relationships in the REFINE object base are based on the names of the objects, we can fill the attributes in as we create each object. When all the objects have been created, we call the necessary housekeeping functions that used to be called after parsing an application from a file, and the application is available.

4.6 Domain-Definition Object Model Development

One of the goals of our work was to abstract implementation-specific details away from the domain expert and Application Specialist. We want the user of the application composition system to be able to work in domain-specific terms to compose applications in that domain. One area where we recognized we could provide an additional level of abstraction was the generation of the domain-specific schema for the database and the source code for the domain primitives. We knew that if we could formally capture the domain-knowledge used to generate the schema and the source code, we could automate the process. The first step was to analyze the type of data needed to produce both the schema and the source code for primitive objects and to create an object model of domain-specific data. We then had to provide tools to transform the domain knowledge into database schemas and source code for domain primitives. A more detailed description of this process follows.

4.6.1 Abstraction of Domain-Oriented Object Models. Our goal here was to produce an object model of object models (a meta-model). Specifically, we wanted to be able to create what we called a domain-definition object model. With this domain-

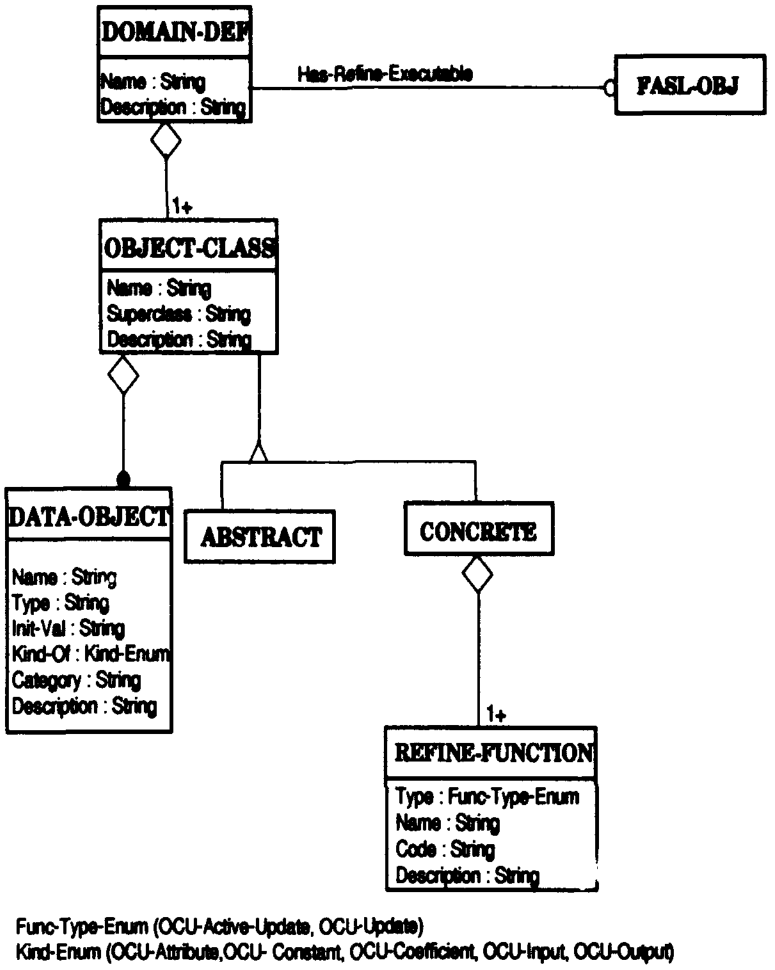


Figure 4.7 Domain-Definition Object Model

definition object model we wanted to be able to capture all the information necessary to define the domain-specific database schema and the primitives for a given application domain. The domain model we produced for the logic circuits domain (Figure 4.1) is an example of a domain model capable of providing that type of domain-specific data. Using Rumbaugh's (29) OMT we analyzed the domain of domain-specific object models to produce the domain-definition object model depicted in Figure 4.7. The logic circuits domain object model depicted in Figure 4.1 is an instance of the type of object model the Domain Engineer is able to describe using the domain-definition object model. A description of the domain-definition object model (Figure 4.7) follows.

1. DOMAIN-DEF – The DOMAIN-DEF class is a container class encapsulating all defined domains. It has a name attribute and a description attribute, both of type string. In our example, the name attribute of our instance of DOMAIN-DEF has the value "Circuits" identifying it as the circuits domain. The container class that acts as a handle for a domain comes from the "Name" attribute of an instance of the DOMAIN-DEF class (refer to the "Circuits" class in Figure 4.1). Instances of the OBJECT-CLASS class in Figure 4.7 represent all the classes below "Circuits" in Figure 4.1. The DOMAIN-DEF class has no association with the Data-Object class or the Refine-Function class, which is how we define attributes and operations respectively. This explains why we implemented "Circuits" as a container class with no attributes or operations. The Domain Engineer can use the optional description attribute of the DOMAIN-DEF class and all other classes in the domain-definition object model to help document the purpose of an object class.

2. FASL-OBJ – The FASL-OBJ class represents instances of REFINE executables. The "Has-Refine-Executable" relationship connects the FASL-OBJ class to the DOMAIN-DEF class. Once a domain has been completely defined and a REFINE executable has been generated, the "Has-Refine-Executable" relationship will connect the DOMAIN-DEF with an instance of the FASL-OBJ class. If a domain exists, but a REFINE executable has not yet been generated, then no instance of the FASL-OBJ class will exist and the "Has-Refine-Executable" relationship will be undefined. The possibility that a domain may or may not have a FASL-OBJ associated with it at any given time explains why the "Has-Refine-Executable" relationship has a zero-to-one multiplicity on the FASL-OBJ class.

3. OBJECT-CLASS – The OBJECT-CLASS class defines the classes in a domain-specific object model. An OBJECT-CLASS has three attributes: name, superclass, and description. In our example in Figure 4.1, the SWITCH, GATE, LED, and MUX classes are all instances of a specialization of OBJECT-CLASS. The GATE instance, for example, would have the following attribute values:

   - name : "gate"
   - superclass : "circuit"

- description : Some description of the class that would aid an Application Specialist in understanding this object's purpose.

Note that the DOMAIN-DEF class contains an "is-composed-of" relationship to one or more instances of the OBJECT-CLASS class. In the circuits example of Figure 4.1, there are fifteen instances of OBJECT-CLASS.

- CIRCUIT-ARTIFACT
- GATE
- SWITCH
- LED
- COMPONENT
- AND-GATE
- OR-GATE
- NAND-GATE
- NOR-GATE
- NOT-GATE
- COUNTER
- DECODER
- MUX
- JK-FLOP-FLOP
- HALF-ADDER

4. DATA-OBJECT – The DATA-OBJECT class defines the attributes of an object class. Note that the "is-composed-of" relationship from OBJECT-CLASS to DATA-OBJECT contains a zero-to-many multiplicity. This allows an instance of OBJECT-CLASS to have zero-to-many attributes defined. The SWITCH OBJECT-CLASS has four instances of DATA-OBJECT associated with it.

    - Delay Attribute
        - Name : "delay"
        - Type : "integer"
        - Init-Val : "0"
        - Kind-Of : OCU-ATTRIBUTE
        - Category : "signal"
        - Description : "descriptive text"
    - Debounced Attribute

  - Name : "debounced"
  - Type : "boolean"
  - Init-Val : "nil"
  - Kind-Of : OCU-ATTRIBUTE
  - Category : "signal"
  - Description : "descriptive text"
- Position Attribute
  - Name : "position"
  - Type : "symbol"
  - Init-Val : on
  - Kind-Of : OCU-ATTRIBUTE
  - Category : "signal"
  - Description : "descriptive text"
- out1 Output
  - Name : "out1"
  - Type : "boolean"
  - Init-Val : "nil"
  - Kind-Of : OCU-OUTPUT
  - Category : "signal"
  - Description : "descriptive text"

The LED OBJECT-CLASS has two instances of DATA-OBJECT associated with it.

- Color Attribute
  - Name : "color"
  - Type : "symbol"
  - Init-Val : red
  - Kind-Of : OCU-ATTRIBUTE
  - Category : "signal"
  - Description : "descriptive text"
- in1 Input
  - Name : "in1"
  - Type : "boolean"
  - Init-Val : "nil"
  - Kind-Of : OCU-INPUT
  - Category : "signal"
  - Description : "descriptive text"

All the object classes depicted in Figure 4.1 are defined similarly.

5. ABSTRACT – All instances of the OBJECT-CLASS class are either ABSTRACT or CONCRETE because of the is-a relationship from OBJECT-CLASS to ABSTRACT and CONCRETE. If an instance of OBJECT-CLASS is an ABSTRACT class, it requires no additional information. Hence, the ABSTRACT class has no attributes or relationships. Using the circuits example again in Figure 4.1, GATE and COMPONENT are both examples of an ABSTRACT OBJECT-CLASS. The attributes shown on GATE and COMPONENT in Figure 4.1 are actually inherited by their subclasses when instances of those subclasses are instantiated. Recall that these are not class attributes, so it is just the attribute description that is inherited, not the attribute value. The reason we need to define the GATE class as an abstract class is because, in our domain, it makes no sense to create an instance of a GATE unless it is further defined as an AND, OR, NAND, NOR, or NOT GATE. Similarly, OBJECT-CLASS is an abstract class within our domain-definition object model depicted in Figure 4.7. It makes no sense to create an OBJECT-CLASS unless it is created as an ABSTRACT or CONCRETE OBJECT-CLASS.

6. CONCRETE – Like ABSTRACT, CONCRETE is a specialization of the abstract class OBJECT-CLASS in the domain-definition object model depicted in Figure 4.7. CONCRETE inherits the attributes of OBJECT-CLASS, just as instances of ABSTRACT do. An instance of the CONCRETE class, however, is composed of one or more instances of the REFINE-FUNCTION class. The reason for this is that all CONCRETE classes in our domain model represent primitives of that domain, and each primitive must have at least one update function. An example of a CONCRETE object in Figure 4.1 is the Counter. Besides the attributes it inherits from its parent classes in the circuits object model, it has the following attributes: the-count, clock, reset, lsb, msb, and max-count. If you look at the REFINE implementation of the counter in Appendix B, you will see that it has one update function. Counter, then, would be associated with only one instance of the REFINE-FUNCTION class. AND-GATE, on the other hand, has two update functions, "AND-GATE-UPDATE" and "AND-GATE-UPDATE2," so it would be associated with two instances of the REFINE-FUNCTION class.

```
format(debug-on, "COUNTER-UPDATE1 on ~s~%", name(counter));

let (clock : boolean = get-import('clock, subsystem, counter),
     reset : boolean = get-import('reset, subsystem, counter))

(if reset then
   COUNTER-THE-COUNT(counter) <- 0
 elseif clock then
   COUNTER-THE-COUNT(counter) <- COUNTER-THE-COUNT(counter) +1
);
(if COUNTER-THE-COUNT(counter) >
        get-coefficient-value(counter, 'max-count) then
   COUNTER-THE-COUNT(counter) <- 0
);
 if COUNTER-THE-COUNT(counter) = 0 then
   set-export(subsystem, counter, 'msb, nil);
   set-export(subsystem, counter, 'lsb, nil)
 elseif COUNTER-THE-COUNT(counter) = 1 then
   set-export(subsystem, counter, 'msb, nil);
   set-export(subsystem, counter, 'lsb, true)
 elseif COUNTER-THE-COUNT(counter) = 2 then
   set-export(subsystem, counter, 'msb, true);
   set-export(subsystem, counter, 'lsb, nil)
 elseif COUNTER-THE-COUNT(counter) = 3 then
   set-export(subsystem, counter, 'msb, true);
   set-export(subsystem, counter, 'lsb, true)
```

Figure 4.8 REFINE Code for Update Function

7. REFINE-FUNCTION – The REFINE-FUNCTION class captures the update functions of a CONCRETE class. To continue using the counter as an example, the single instance of the REFINE-FUNCTION class associated with counter would be:
   - Type : "OCU-Active-Update"
   - Name : "COUNTER-UPDATE1"
   - Code : (See Figure 4.8)
   - Description : "descriptive text"

Figure 4.9 is an instance diagram of the Domain-Definition object model instantiated for the logic circuits domain pictured in Figure 4.1. Take note that this is an instance diagram of only a small subset of the object model depicted in Figure 4.1 for purposes of illustration. Notice that the Domain-Definition instance in Figure 4.9 has the name value "CIRCUITS" which corresponds to the CIRCUITS container class near the top of Figure 4.1. In the ICO-OBJECT-CLASSES relationship of the DOMAIN-Definition instance in Figure 4.9 are four CONCRETE OBJECT-CLASS instances and two ABSTRACT

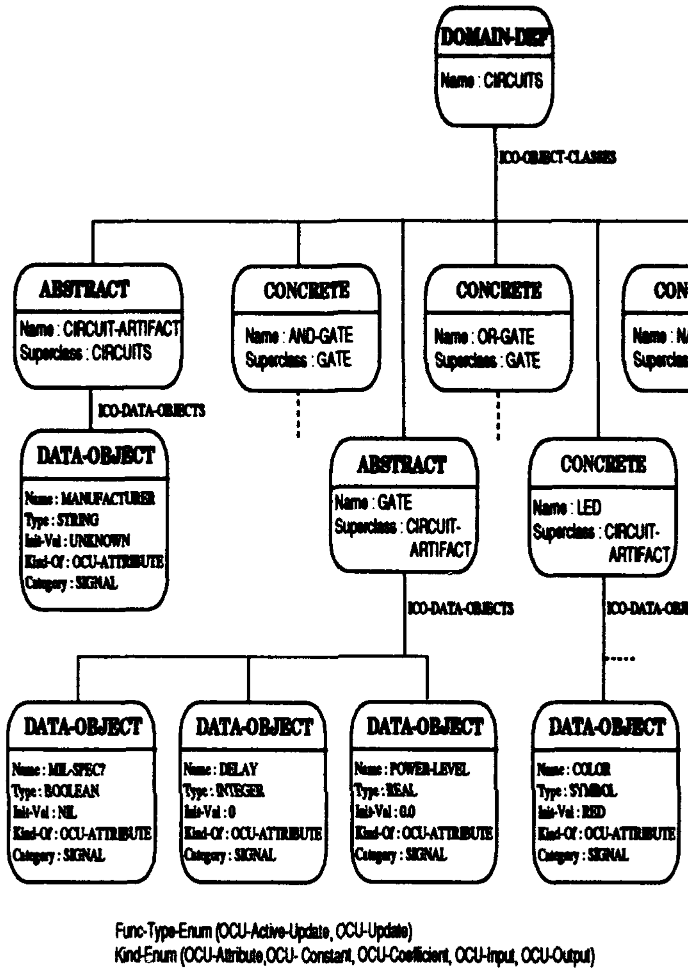


Figure 4.9 Domain-Definition Instance Diagram for Logic Circuits Domain Subset

OBJECT-CLASS instances. Each one of these can be traced back to CONCRETE and ABSTRACT OBJECT-CLASSES respectively on the object model in Figure 4.1. For example, look at the CONCRETE instance on Figure 4.9 with the name value LED and the superclass value Circuit-Artifact. Note that it has a DATA-OBJECT in its ICO-DATA-OBJECTS relationship, and that DATA-OBJECT is an OCU-ATTRIBUTE with the name COLOR. Now refer back to Figure 4.1 and find the CONCRETE class labeled LED. Its superclass is Circuit-Artifact and it has a color attribute. Recall that the instance diagram in Figure 4.9 only represents a small subset of the whole instance diagram for Figure 4.1, but the reader should be able to trace every element in Figure 4.9 back to a corresponding element pictured in Figure 4.1.

Satisfied that the domain-definition object model in Figure 4.7 was capable of capturing the domain knowledge in a domain model like that in Figure 4.1, we implemented the domain-definition object model as a database schema in the OODBMS. The following sections describe the process of populating the database with a description of an application domain, and generating a database schema and REFINE source code for primitive objects from that data.

4.6.2 Populating Database with Domain Knowledge. One of the tools available with the ITASCA OODBMS is its Active Data Editor (ADE™). It allows the user of the OODBMS to create instances of the classes in the domain-definition object model to describe a given application domain. We found from experience that the object model we developed was very good for defining the inheritance relationships, showing abstract versus concrete, and identifying attributes, but it got busy very quickly when we tried to put a lot of detail on it. For that reason, we developed some data collection forms to use in conjunction with the object model to make it simpler for the Domain Engineer to input data into the database. Appendix A has a description of each data collection form, as well as example completed forms for all the primitives in the logic circuits domain. These are just tools to help organize the data, but we found it very helpful to have these all filled out before attempting to enter the data into the database. Appendix D contains the scenario of defining the logic circuits domain using the domain-definition object model and the ITASCA ADE.

Once the Domain Engineer enters the domain knowledge into the database, as we did in Appendix D, we have access to enough domain knowledge to produce a database schema capable of storing instances of domain primitives. The domain knowledge, coupled with some knowledge of the structure of the application composer's source code, enables us to generate REFINE source code for the primitive objects of the domain. The next two sections describe these two capabilities.

4.6.3 Automatic Generation of Database Schema. With knowledge of the domain now captured in the database, we needed to write a set of database methods to use that knowledge to generate a database schema for storing domain-specific data. The database

```
(def-class CLASS-NAME
:superclasses (SUPERCLASS-NAME )
:attributes (

  (ATTRIBUTE-NAME
:domain ATTRIBUTE-TYPE
:init   INITIAL-VALUE  )

  (ATTRIBUTE-NAME
:domain ATTRIBUTE-TYPE
:init   INITIAL-VALUE  )

  (ATTRIBUTE-NAME
:domain ATTRIBUTE-TYPE
:init   INITIAL-VALUE  )
)
:class-attributes ())
```

Figure 4.10 ITASCA Schema Template

```
(def-class CIRCUITS
:superclasses (OCU-PRIMITIVE )
:attributes ()
:class-attributes ())
```

Figure 4.11 ITASCA Class Definition of CIRCUITS Class

methods serve as a template for the commands needed to generate a database schema. The template for defining a class in the database schema is shown in Figure 4.10.

There can be from zero to many attributes defined for a class. In the above case three attributes are defined. We do not currently use class attributes, so the class attributes field will always be empty. The words in upper-case represent the "place holders" in the template that are filled with instance data from the DOMAIN-DEF object model.

To continue with the logic circuits domain, the instance of the DOMAIN-DEF class in Figure 4.9 has the name CIRCUITS. Recall too that instances of DOMAIN-DEF serve as our container class and inherit from the architecture primitive. The code generated from this instance data is in Figure 4.11.

The algorithm the database method follows is to then visit each object class in the ico-object-classes relationship and produce the code for each of these object classes. Some object classes have an ico-data-objects relationship. So, while visiting those object

```
(def-class CIRCUITS
:superclasses (GATE )
:attributes (

  (DELAY
:domain INTEGER
:init   0  )

  (MIL_SPEC?
:domain BOOLEAN
:init   NIL  )

  (POWER_LEVEL
:domain REAL
:init   0.0  )
)
:class-attributes ())
```

Figure 4.12 ITASCA Class Definition of the GATE Class

classes, all DATA-OBJECTS in the ico-data-objects relationship are visited to produce the appropriate code. The code for the instance of ABSTRACT-CLASS named GATE is in Figure 4.12.

Note that the 3 DATA-OBJECTS in the ico-data-objects relationship were visited and their corresponding code was produced. The ITASCA code to generate the database schema for the entire logic circuits domain is in Appendix B.

4.6.4 Automatic Generation of REFINE *Source Code.* We also needed to write a set of methods to use the domain knowledge captured in the database to produce REFINE source code for the primitive objects of the domain. The same general process used to generate the database schema is also used to generate the REFINE source code. The methods encapsulate the knowledge that represents a template for defining domain primitives in REFINE source code. The instances of data in the DOMAIN-DEF object model are visited in the proper order to fill in the blanks in the template. The REFINE source code generated by these methods can be found in Section B.2. This code can be compiled by REFINE, and be used in the REFINE Object Base to define the logic circuits domain for the application composer.

4.7 Incorporation of Concurrent Research Results

There were a number of concurrent research efforts on the domain-oriented application composition environment that we needed to stay abreast of, especially in the area of improving the visual interface (8) and support for an application executive (11, 36).

4.7.1 Changes for the Visual Interface.

Cossentine (8) completely changed the face of Architect's visual interface by introducing bitmapped icons for representing domain primitive objects. Our effort in supporting these changes were:

1. Support the storage of the bitmapped icon definitions within the database, allowing them to be read directly from the database rather than separate files.
2. Allow the Domain Engineer to associate the name of bitmapped-icons to correspond to primitive definitions in his domain model.
3. Ensure our source code generation method provided the necessary code to read in the bitmap information from the database when the domain model gets loaded from the database to the Architect object base in REFINE.

4.7.2 Addition of Application Executive Services.

Welgan (36) researched the ability to have application executives support different operating environments, to include discrete event simulation and real-time applications. Although we do not support any of the new application areas he worked on, the domain model representation for the OCU model was modified to support his research. Specifically, imports and exports from lower level subsystems were moved up to the highest level subsystem. These changes were application specific, so they did not affect our abstract OCU database schema. However, our mappings for correctly saving and restoring these objects had to be modified to support these changes.

4.8 Conclusion

In this chapter we introduced the logic circuits domain as the validating domain for our work. We then developed an object model of the logic circuits domain using Rumbaugh's Object Modeling Technique. We presented our object model of the OCU

Architecture and introduced the concept of the Domain-Definition Object Model. By populating the Domain-Definition Object Model with knowledge of a domain, we showed that we could

1. Automatically generate in the technology base, the database schema for the newly defined domain, and
2. Automatically generate the REFINE source code to define the domain in the Domain-Oriented Application Composition Environment.

Finally, we described the integration of the ITASCA OODBMS version of the technology base with the original file-based version of Architect. In the next chapter we describe the testing we conducted to validate our work. We describe the stand-alone testing we did for each of the functional components and the integration testing we did to ensure all the pieces worked together properly.

V. Validation and Analysis

5.1 Introduction

The validation domain used for AVSI with database capabilities was digital circuits, the same domain used by Anderson and Randour for Architect. This chapter discusses the validation of our research using this domain and presents an assessment of AVSI with database capability.

Since we did our development for this research in parallel with each other, each branch was tested separately, followed by a complete test of the entire project. The following sections summarize the testing methodology and results.

5.2 Testing of AVSI with Database Capabilities

As noted before, we developed the database capabilities for AVSI using a rapid-prototyping, evolutionary development approach. The goal was to be able to store and retrieve applications composed in AVSI sessions. Testing, in a rapid-prototyping approach, is done in parallel and in conjunction with development.

We used the following test objectives:

1. The ability to successfully store a composed application in the database.
   (a) Verify each object in the REFINE object base that is supposed to be saved in the database gets instantiated.
   (b) Verify the attribute values of each object created in the database.
   (c) Verify the structural relationships of the application stored in the database.
2. The ability to successfully load an application stored in the database into the REFINE object base.
   (a) Verify that each object stored in the database gets instantiated in the REFINE object base.
   (b) Verify the attribute values of each object created in the REFINE object base.

(c) Verify the structural relationships of the application in the REFINE object base.

(d) Ensure the application loaded into the REFINE object base correctly executes.

(e) Store the application loaded from the database to a file, and compare it with the file created when the application was originally composed.

Before beginning development of code, we first had to instantiate the ITASCA schema developed for the circuits domain in Figure 4.1. We input this information directly into ITASCA using the schema editor tool provided.

The first phase of development and testing involved saving an application into the database. The goal was to load an entire application from the file-based technology base and save it to the database. After each attempt, we walked through the instances contained in the database, verifying that all necessary objects existed and that each contained the correct data. This was accomplished incrementally. First, we attempted to just create the necessary objects; next, we copied and verified the object's attribute values; finally, we added the structure by updating and verifying the relationships between objects. After each iteration was verified, we deleted all of the objects in the database to begin the next step. Upon complete verification of the database instances, this phase of development was almost complete. It could not be completely verified until the next phase was also completed.

The next phase of development and testing built on the first phase; now we must be able to load the saved application from the database back into the REFINE object base during an AVSI session. Again, this was done incrementally, first building the objects, then updating the attributes. We manually verified these steps using the object browser feature of REFINE. After passing the manual verification, we verified that the application would execute properly within the AVSI session. The final verification involved saving the application loaded from the database back out to the file-based technology base under a different file name. We compared the original source file, used in one AVSI session to populate the database, with the file generated under a separate session and saved out to file. We repeated this final verification for each of the validated applications contained in the file-based technology base; after completing this verification, the database technology base contained the same information as the file-based version.

5.3 Testing of Database Methods

We developed two sets of database methods that traverse the DOMAIN-DEF structures in the ITASCA database, extracting domain knowledge to produce domain and schema definitions for the Domain-Oriented Application Composition Environment. Before testing could start, we had to instantiate the ITASCA schema for the DOMAIN-DEF object model. We did this by using the interactive ITASCA Schema Editor. We then populated the database with the definition of the logic circuits domain. We used the ITASCA Active Data Editor to do this, just as a Domain Specialist would be expected to do when defining a new domain. Once we had the knowledge of the logic circuits domain loaded into the database, we could test the two sets of methods we developed.

We used the following test objectives:

1. The ability to represent domain knowledge with instances of the Domain-Definition object model.
2. The ability to automatically generate a database schema definition in the ITASCA database from instances of the Domain-Definition object model.
3. The ability to automatically generate REFINE source code to load a domain definition into the REFINE Object Base from instances of the Domain-Definition object model.

5.3.1 Testing Methods that Generate Database Schemas. The first step in testing the schema generation methods was to manually derive the set of ITASCA commands needed to create a database schema for the logic circuits domain. We verified that our set of commands was syntactically correct by using them to instantiate the schema interactively through the ITASCA Schema Editor. At this point we manually verified that the schema we generated would be sufficient to store instances of the domain primitives. We then executed the set of methods to automatically generate the database schema for the logic circuits domain, sending the output to a file. Next we compared the schema generation commands produced by the methods with the ones we had produced manually. We deferred the functional testing of the schema until integration testing when the database load and retrieve capabilities would also be available.

5.3.2 Testing Methods that Generate REFINE *Source Code.* We took a similar approach to testing this set of methods. In this case, we already had source code to generate the logic circuits domain from the baseline Domain-Oriented Application Composition System. We executed the set of methods to automatically generate the REFINE source code for the logic circuits domain, sending the output to a file. Although the format was somewhat different (indentations, line spacing, etc), we could verify that the automatically generated code would produce the same results in the REFINE object base as the baseline code. We further tested our code by starting an Architect session and loading the domain from our file instead of the baseline files. We then used Architect to compose and execute applications in the logic circuits domain to verify that Architect still worked properly.

5.4 Testing of Integrated System

Now that each piece of the project had passed stand-alone testing, it was time to put it all together. Before integration testing began, we deleted all instance data and schema definitions generated from stand-alone testing described in Section 5.2 and Section 5.3. We now had a clean environment for integration testing.

We first played the role of the Domain Engineer, using the ITASCA ADE to enter the circuits domain definition. We then initialized an AVSI session in REFINE, using our database version. We ran the subprogram provided for the Domain Engineer which generates the ITASCA schema for a domain and also generates and compiles the REFINE source code for the AVSI domain model. If no errors are encountered, the database schema is committed and the REFINE executable is saved in the database. When this was successful, we manually verified the schema using the schema editor and verified the source code by examining the file. We also intentionally introduced errors to ensure that our error detection functioned properly.

Having successfully completed and verified the Domain Engineer function, we shut AVSI down. We now needed to verify that the products created could be used by an Application Specialist. We started a new AVSI session, and composed a new application. After verifying the application could execute properly, we saved a copy of it to both the file system and the database. We would use the file system version later for verification.

Once again we shut AVSI down and started a new session. We reloaded our saved application from the database and verified its execution within the AVSI environment. Once execution was verified, we saved the application to a new file and compared this to the previously saved file. Having successfully accomplished this, we began the process of loading in each saved application from the file system and saving them to the database. In another AVSI session, we reloaded each application from the database, verified its execution, saved it to a different file, and compared the two file versions. When this was completed, we again had an exact copy of the file-based technology base saved in the database.

5.5 Validating Support for Concurrent Research Results

In order to support concurrent research, we developed our changes to Architect in isolation starting from a known baseline. As each successive baseline was created and verified by other researchers, we incorporated and tested all of their changes into our code. This kept our code in line with the latest baseline during our development. Each time a new baseline was created, we worked closely with its developer to test any new functionality added, as well as to verify no functionality was lost.

5.6 Conclusion

In this chapter we discussed the stand-alone and integration testing of our work. We presented the test objectives for each functional component and described the tests to ensure we had met those objectives. Finally, we described the integration testing we did to ensure all the functional components worked properly together. The results of our testing verified that we had accomplished the goals we had set for this research. In the next chapter we discuss the conclusions we draw from our research and make recommendations for future research.

VI. Conclusions and Recommendations

The objective of this research was to demonstrate that an OODBMS could be used as a Modeling Library for managing complex, object-oriented software simulation components. We wanted to show that integrating an object-oriented database with a domain-oriented application composition system would create a software engineering environment that promotes reuse of software artifacts stored in a domain-oriented technology base. We demonstrated this by integrating the ITASCA OODBMS with the Architect domain-oriented application composition system. In this integrated environment, the OODBMS serves as the central repository (technology base) for domain-specific artifacts such as primitive domain components, subsystems, visualization data, and composed applications.

We first examined the current literature on object-oriented database systems in an attempt to define what capabilities they could provide, and how they could aid us in reaching our goal. Some of the OODBMS capabilities we felt would be most important for integrating Architect into this software engineering environment were a dynamic schema editing capability to aid in rapid prototyping, support for a Common Lisp environment to be compatible with Architect, support for versioning to manage multiple versions of the same data, and a distributed database capability to allow multiple sites to access the data simultaneously. We selected the ITASCA OODBMS because we felt it provided the best mix of these capabilities for our new software engineering environment.

The Architect system we began with functioned in a stand-alone environment (refer to Figure 3.2). Figure 6.1 shows the environment that Architect operates in as a result of this thesis. Note that all of the file system interfaces on the left-hand side of the figure are simply what existed before our integration of Architect with an OODBMS. None of the existing file-based functionality was removed, and the added database functionality is shown on the right-hand side of Figure 6.1.

A new load file is provided for the Application Specialist to take advantage of the database capabilities we have added. DIALECT and INTERVISTA are automatically loaded just as they were with the file-based system; however, the Architect software and

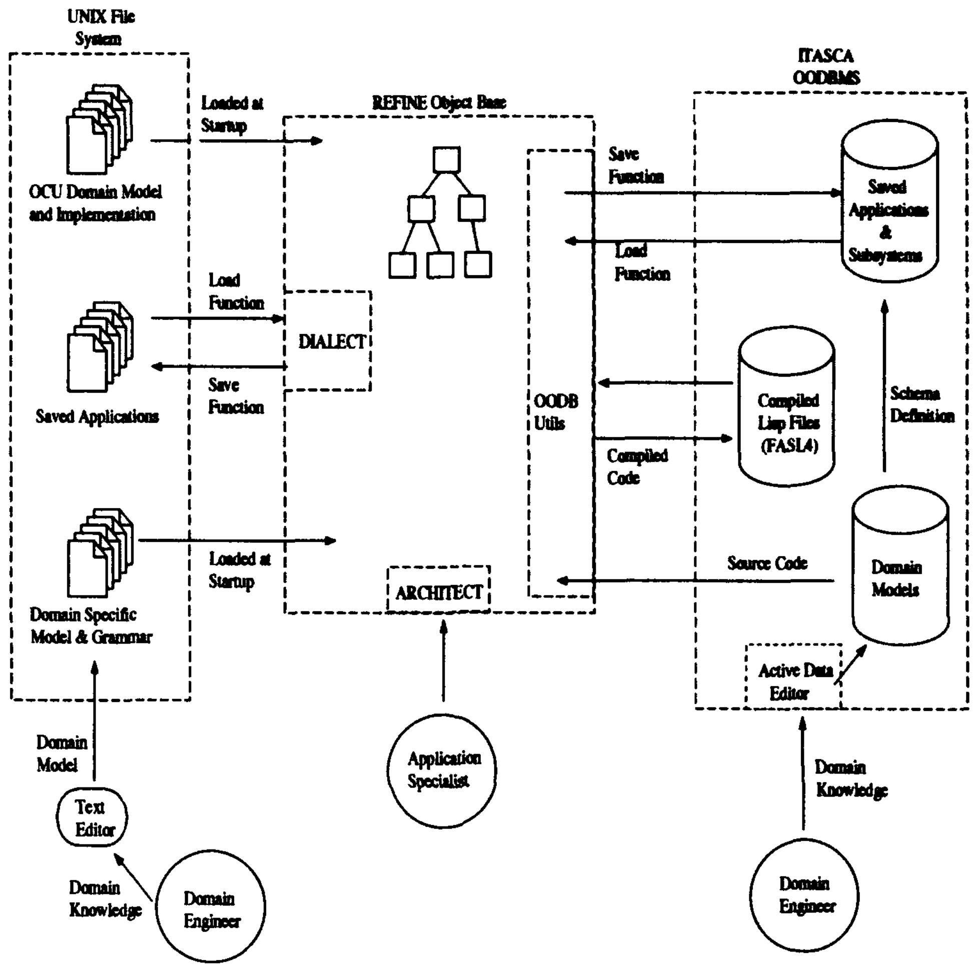


Figure 6.1 Big Picture of the New Architect Environment

OCU architecture model are now loaded directly from the database. Unlike the previous implementation, no domain-specific information is loaded upon startup. The first time the Application Specialist refers to a domain, that domain's grammar, domain model, and primitive descriptions are generated by the transformation methods and loaded in to the REFINE object base. Note that the grammar is loaded to maintain the previous file-based functionality only; the database implementation eliminates the requirement for the domain-specific grammar. When the primitive descriptions are loaded into the REFINE object base, the icon bitmap information is also loaded from the database, eliminating the need for a visual grammar.

The menu for loading and saving applications now contains options to specify either the database or the file system. If the Application Specialist loads from the database version of the technology base and uses the database for storage and retrieval, then the requirement for running REFINE from a specified directory structure is eliminated. This also ensures that all user's are using the same version of the software, as it is now centrally located. In the file-based version, many copies of the source code can exist in many user's directories, making configuration management much more difficult.

We developed a domain-definition meta-model using Rumbaugh's object modeling technique (29) to define a class of domain-oriented object models. We instantiated this meta-model as a schema in the ITASCA OODBMS, and populated the database with the description of an object model we developed for the logic circuits domain. With this meta-model resident in the database, the Domain Engineer can now enter the results of his domain analysis directly into the database using the ADE tool provided with ITASCA. All of the restrictive, implementation-dependent naming conventions have been removed, as well as the requirement to define the domain in a target implementation language.

Having the logic circuits domain formally defined in the database allowed us to write database methods to transform the logic circuits object model into other formal representations. We wrote two such sets of database methods to operate on instances of the domain-definition meta-model. One set of methods encapsulates the knowledge necessary to transform domain-definition data into the REFINE source code used to instantiate a domain in the REFINE object base. These methods automatically generate the implementation-dependent naming conventions of the target language. The other set of methods transform the same domain-definition data into the ITASCA schema-creation commands needed to store domain artifacts in the ITASCA OODBMS. As the technology base of domains and architectures grows, so does the importance of having automatic transformations from a domain description to other formal representations required by client applications.

6.1 Conclusions

Using an OODBMS for the technology base in a domain-oriented application composition environment (and other object-based software engineering tools) provides distinct advantages over using a file-based technology base. We successfully demonstrated this by integrating the ITASCA Distributed OODBMS with the Architect Domain-Oriented Application Composition System. The OODBMS technology base not only provides all the functionality previously available from the Architect file-based technology base, but provides significant additional capabilities as well.

The greatest benefit of our prototype domain-oriented application composition environment is the ability to maintain domain data in object-model form throughout its life-cycle. The Domain Engineer can input the object-model for a domain directly into our prototype domain-oriented application composition environment, eliminating the need to artificially flatten the object model into a structure compatible with a file-based technology base. For Architect, this eliminated the need for the domain-specific grammar and the visual-system grammar which were used to parse file-based descriptions of domain data between the file system and the object base.

We also found that, for a given domain, there is a minimal set of data reflected in an object model that is duplicated many times when the object model is flattened to conform to a file-based format. By keeping the domain data encapsulated in one location in the database, we eliminate the duplication of data, and we can better protect it from corruption. Changes to domain data are now centralized, and we are guaranteed that all applications accessing the database are working with the same domain definition.

The OODBMS technology base greatly increases the reusability of not only the domain-oriented artifacts stored in the database, but of the domain and architecture models used to implement a domain-oriented application composition environment as well.

The OODBMS environment is much more conducive to exploiting the power offered by object modeling and model-based software development. We found it to be a very natural environment in which to develop a meta-model for domain models, enabling us to define domains in an implementation-independent way. By abstracting a class of models into a meta-model and implementing that meta-model in an OODBMS we can significantly reduce the effort required to generate multiple versions of the same or similar data. Our development and implementation of a domain-definition meta-model demonstrated this capability.

6.2 Recommendations for Further Research

- Develop a Meta-Model of Architecture Models

  Just as we developed a meta-model for a class of domain models, a meta-model describing a class of architecture models should be investigated. Architect is based on the OCU architecture model, but if a common formal description of architecture models existed, it would ease the transformation of reusable software artifacts between architectures, thereby improving reusability. Gool (11) conducted some research into alternative architectures for systems like Architect that might be helpful.

- Further Abstract the Transformation Methods for the Domain-Definition Model

  Currently the methods in the domain-definition meta-model encapsulate knowledge of the target environment. In the case of the methods used to generate the database schema, the methods encapsulate knowledge of the grammar for the ITASCA schema-generation code. In the case of the methods used to generate the REFINE source code, knowledge of the REFINE programming language is encapsulated in the methods. These transformation methods may be further abstracted to allow a formal description of the target environment be an input to the method. Instead of embedding implementation-specific details in the methods, generalize the methods to take as input all the affected models and perform the transformations based solely on inputs. This way, a new set of methods won't have to be written for each environment wishing to access reusable artifacts from a domain.

- Provide Additional Functionality for Meta-Models

  Currently, there are very limited syntactic, semantic, and consistency checks done when the domain-definition meta-model is populated with the definition of a domain.

Methods could be written to perform these types of checks for the Domain and Software Engineers.

- Investigate using the OODBMS for a dynamic simulation environment

  We have concentrated on the static portion of the J-MASS modeling library in our research. The capability to execute simulations of applications in the OODBMS should be investigated. Halloran (12) did some preliminary work in this area.

- Incorporate Other Domains and Architectures

  We attempted to make our work scalable by encapsulating domain and architecture data separately, and relating them through inheritance. The introduction of more architectures (11) and more domains like the digital signal processing domain (34) is needed to further validate these capabilities. As more domains and architectures become available, additional research opportunities arise, such as the sharing of domain artifacts between domains and architectures.

- Explore Versioning and Schema Evolution Capabilities of OODBMS

  The Versioning and Schema Evolution Capabilities offered by the OODBMS could be very beneficial to both the end-user Application Specialist working with different versions of domain data, and the Domain and Software Engineers developing new architectures and domain models.

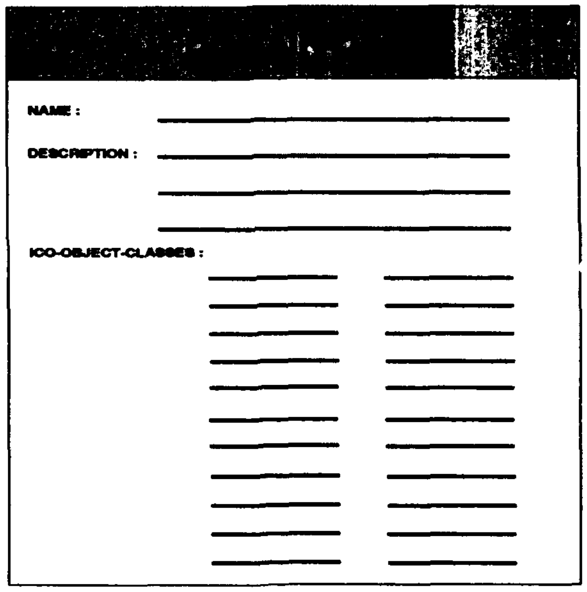
6.3 Final Comments

The domain-oriented application composition environment described in this thesis is a significant step in the right direction toward developing a modeling library for J-MASS. It also serves as a proof of concept for some uses of object-oriented database management systems in software development and composition systems. It is no longer necessary to corrupt object-oriented data into a flattened structure in order to store it in a technology base. By storing object-oriented data in its natural object-oriented form, you maintain those characteristics that led you to use the object paradigm in the first place! With methods stored in the database, new views of abstract data only require the writing of a new method. Once written, it resides with the data, and becomes a tool available to many users. The isolation of data from specific implementations greatly improves the productivity of software engineering professionals as well as application specialists. Using an OODBMS as the technology base for a set of object-based tools is a natural progression in the effort to capitalize on the benefits offered by the object paradigm.

Appendix A. Description of Technology Base Primitives for Logic Circuits Domain

This appendix contains a brief description of the data collection forms used to capture domain knowledge for a given application domain. Having the data in this form makes it easier for the domain specialist to enter the data into the ITASCA OODBMS. Section A.2 has all the forms that were filled out to define the logic circuits domain. Appendix B lists all the REFINE code that is generated from this data by the ITASCA database methods. The code is used by the AVSI/Architect system to define the primitives of the logic circuits domain.

A.1 Description of Data Collection Forms



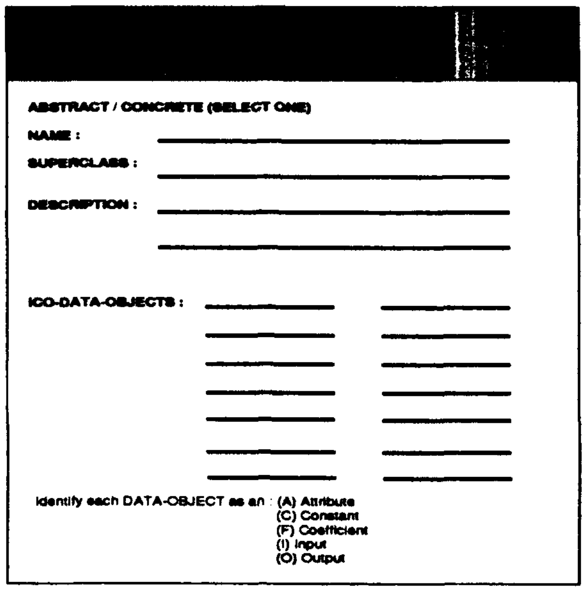
NAME :

DESCRIPTION :

ICO-OBJECT-CLASSES :

Figure A.1 Data Collection Form for DOMAIN-DEF Class

The contents of the Name field on this form are used to name an application domain. Figure A.9, the completed DOMAIN-DEF form for the logic circuits domain, has the value "circuits" in the name field. The corresponding object class created as a result of this data is the "circuits" abstract class near the top of Figure 4.1. You can also see from the Domain-Definition Object Model illustrated in Figure 4.7, that the DOMAIN-DEF class has an is-composed-of relationship with the OBJECT-CLASS class. The completed DOMAIN-DEF form (Figure A.9) for the circuits domain lists all the object classes needed to define the logic circuits domain. This can be verified by comparing Figure A.9 to Figure 4.1. The description field is optional on this and all other data collection forms.



ABSTRACT / CONCRETE (SELECT ONE)

NAME :

SUPERCLASS :

DESCRIPTION :

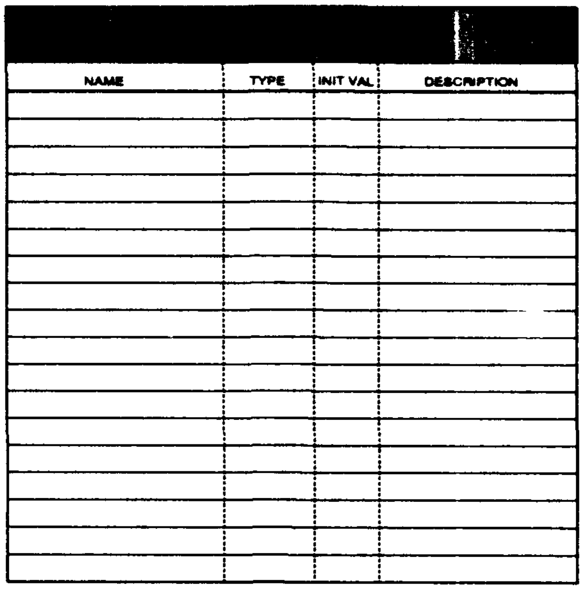
ICO-DATA-OBJECTS :

Identify each DATA-OBJECT as an : (A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

Figure A.2 Data Collection Form for OBJECT-CLASS Class

The Object-Class data collection form is used to describe all the object classes in an application domain. In the case of the logic circuits domain, there are 15 of these forms, one for each object class below "circuits" in Figure 4.1. The logic circuits forms are presented from Figure A.10 through Figure A.24. The form provides for a way to identify whether a class is abstract or concrete. The name field provides the name of the object class. The superclass field is used to describe the hierarchy depicted in Figure 4.1. Note that the data collection form in Figure A.12 which defines the GATE class, lists Circuit-Artifact as its superclass. Figure 4.1, consequently, shows Circuit-Artifact as the superclass of the GATE class.

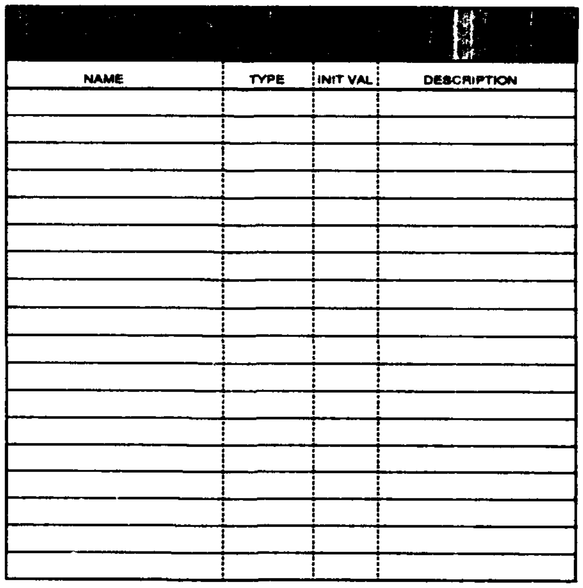
Recall from the Domain-Definition Object Model in Figure 4.7 that an OBJECT-CLASS class has an is-composed-of relationship with the DATA-OBJECT class. The OBJECT-CLASS form has a place to identify those DATA-OBJECTs that are part of this relationship. Each DATA-OBJECT must be further defined as an attribute, constant, coefficient, input, or output using the legend at the bottom of the form.



NAME	TYPE	INIT VAL	DESCRIPTION

Figure A.3 Data Collection Form for OCU Attributes

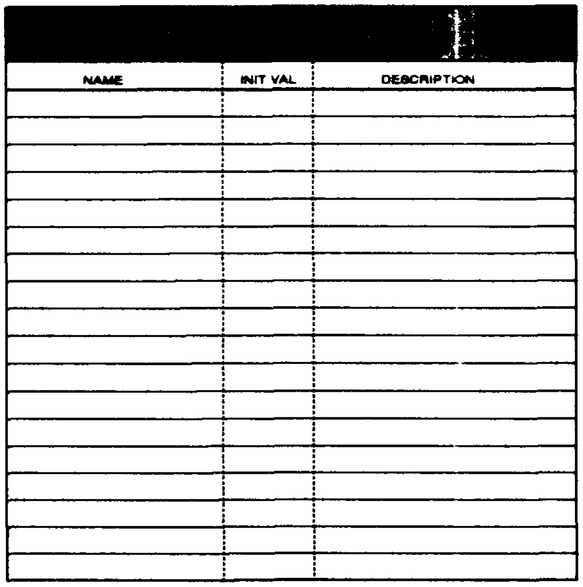
The OCU-ATTRIBUTES form provides a space for each attribute's name, its type (i.e., string, symbol, real, etc), an initial value, and an optional description.



NAME	TYPE	INIT VAL	DESCRIPTION

Figure A.4 Data Collection Form for OCU Constants

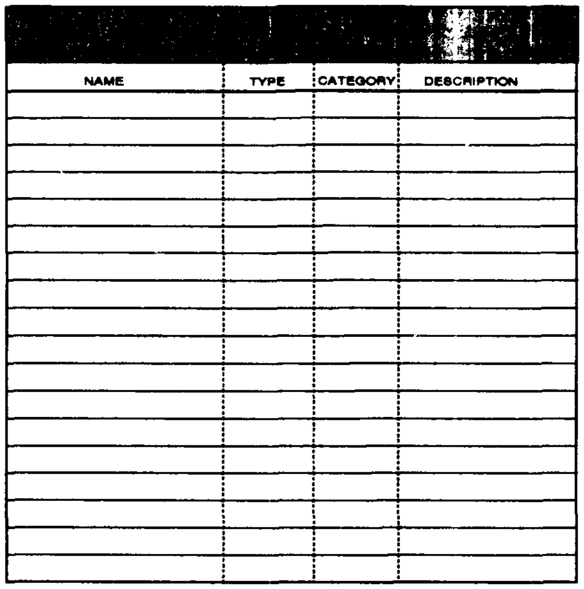
The OCU-CONSTANTS form provides a space for each attribute's name, its type (i.e., string, symbol, real, etc), an initial value, and an optional description.



NAME	INIT VAL	DESCRIPTION

Figure A.5 Data Collection Form for OCU Coefficients

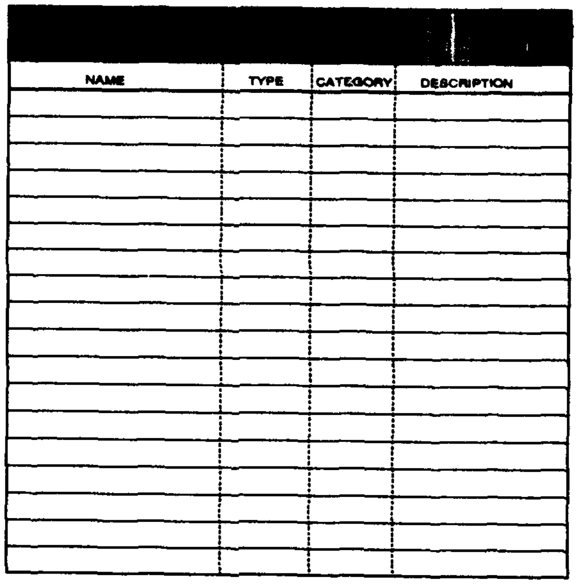
The OCU-COEFFICIENTS form provides a space for each coefficient's name, initial value and optional description.



NAME	TYPE	CATEGORY	DESCRIPTION

Figure A.6 Data Collection Form for OCU Inputs

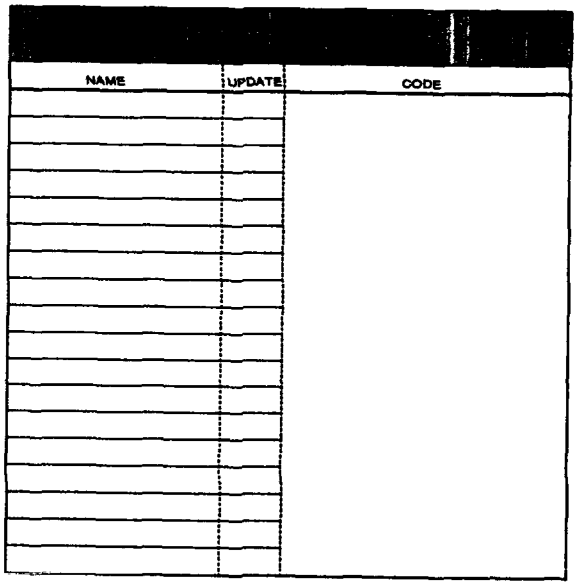
The OCU-INPUTS form provides a space for each input's name, its type (i.e., string, symbol, real, etc), and its category, as well as an optional description.



NAME	TYPE	CATEGORY	DESCRIPTION

Figure A.7 Data Collection Form for OCU Outputs

The OCU-OUTPUTS form provides a space for each output's name, its type (i.e., string, symbol, real, etc), its category, and an optional description.

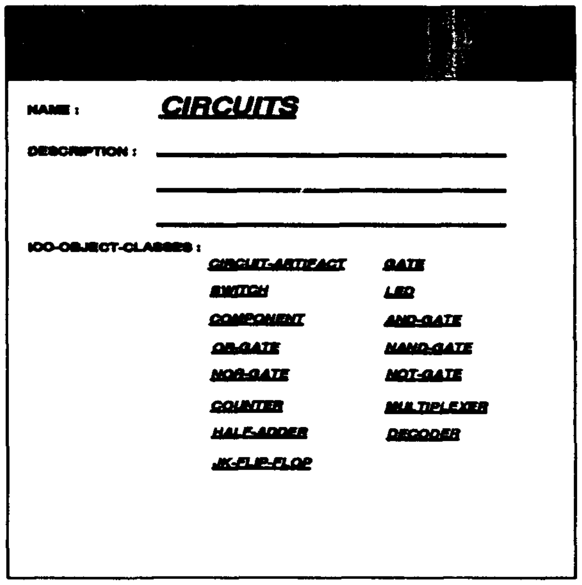


NAME	UPDATE	CODE

Figure A.8 Data Collection Form for OCU Functions

The REFINE-FUNCTION data collection form provides space for each function's name. The UPDATE space allows the domain specialist to identify whether a function is to be used as a function's update function or not. The actual REFINE code can be written in the block provided.

A.2 Completed Data Collection Forms for the Logic Circuits Domain



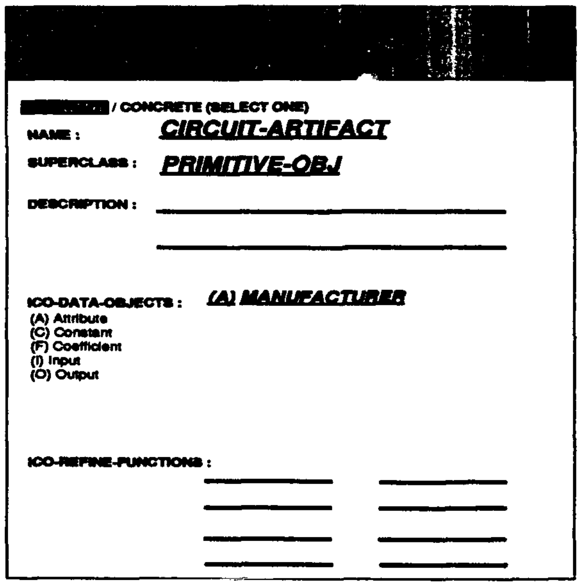
NAME : *CIRCUITS*

DESCRIPTION :

ICO-OBJECT-CLASSES :

CIRCUIT-ARTIFACT	*GATE*
SWITCH	*LED*
COMPONENT	*AND-GATE*
OR-GATE	*NAND-GATE*
NOR-GATE	*NOT-GATE*
COUNTER	*MULTIPLEXER*
HALF-ADDER	*DECODER*
JK-FLIP-FLOP	

Figure A.9 Completed Data Collection Form for DOMAIN-DEF



[redacted] / CONCRETE (SELECT ONE)

NAME : *CIRCUIT-ARTIFACT*

SUPERCLASS : *PRIMITIVE-OBJ*

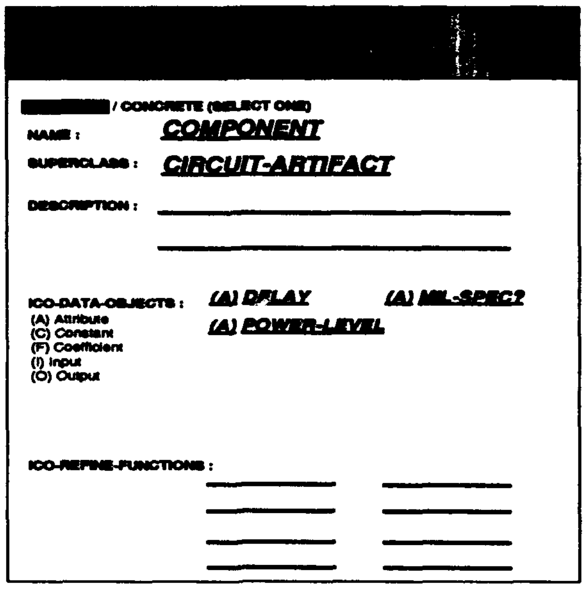
DESCRIPTION :

ICO-DATA-OBJECTS : *(A) MANUFACTURER*

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS :

Figure A.10 Completed Data Collection Form for Circuit-Artifact OBJECT-CLASS


[redacted] / CONCRETE (SELECT ONE)

NAME : *COMPONENT*

SUPERCLASS : *CIRCUIT-ARTIFACT*

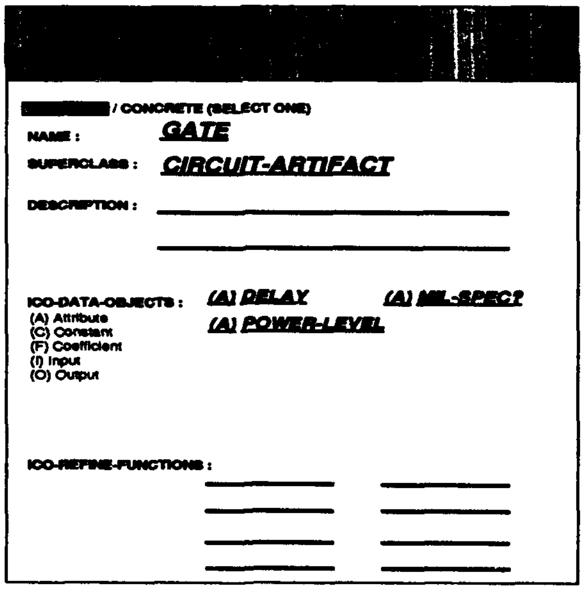
DESCRIPTION : ______________________

ICO-DATA-OBJECTS : *(A) DELAY* *(A) MIL-SPEC?* *(A) POWER-LEVEL*

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS :

Figure A.   Completed Data Collection Form for Component OBJECT-CLASS


[redacted] / CONCRETE (SELECT ONE)

NAME : *GATE*

SUPERCLASS : *CIRCUIT-ARTIFACT*

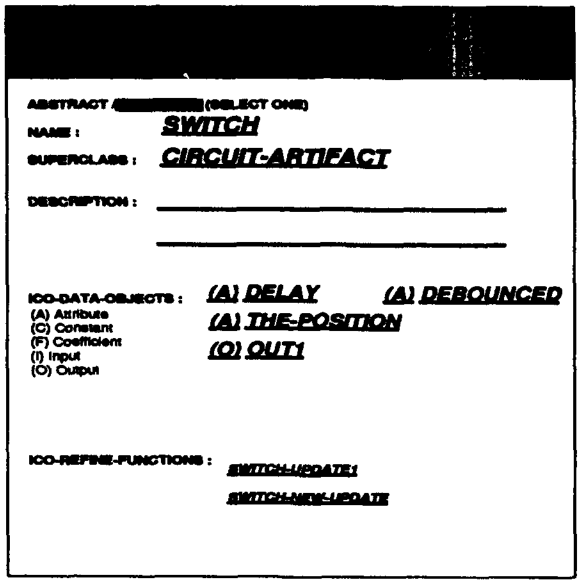
DESCRIPTION : ______________________

ICO-DATA-OBJECTS : *(A) DELAY* *(A) MIL-SPEC?* *(A) POWER-LEVEL*

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS :

Figure A.12 Completed Data Collection Form for Gate OBJECT-CLASS



ABSTRACT / [illegible] (SELECT ONE)

NAME : SWITCH

SUPERCLASS : CIRCUIT-ARTIFACT

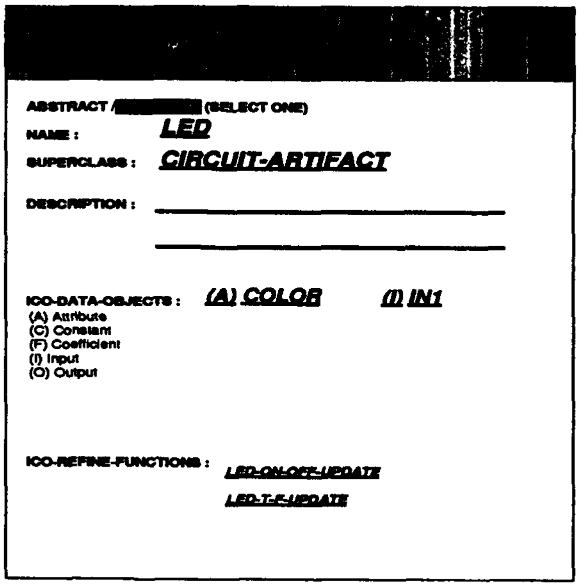
DESCRIPTION : ____________________

ICO-DATA-OBJECTS : (A) DELAY (A) DEBOUNCED (A) THE-POSITION (O) OUT1

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS : SWITCH-UPDATE1 SWITCH-NEW-UPDATE

Figure A.13 Completed Data Collection Form for Switch OBJECT-CLASS



ABSTRACT / [illegible] (SELECT ONE)

NAME : LED

SUPERCLASS : CIRCUIT-ARTIFACT

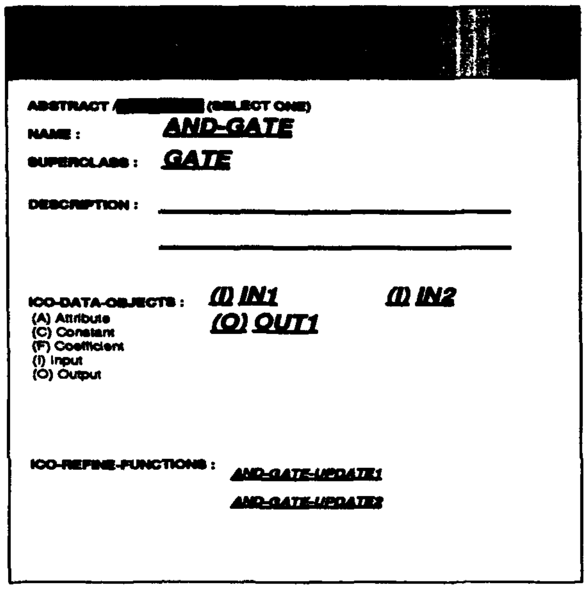
DESCRIPTION : ____________________

ICO-DATA-OBJECTS : (A) COLOR (I) IN1

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS : LED-ON-OFF-UPDATE LED-T-F-UPDATE

Figure A.14 Completed Data Collection Form for LED OBJECT-CLASS



ABSTRACT / [illegible] (SELECT ONE)

NAME : AND-GATE

SUPERCLASS : GATE

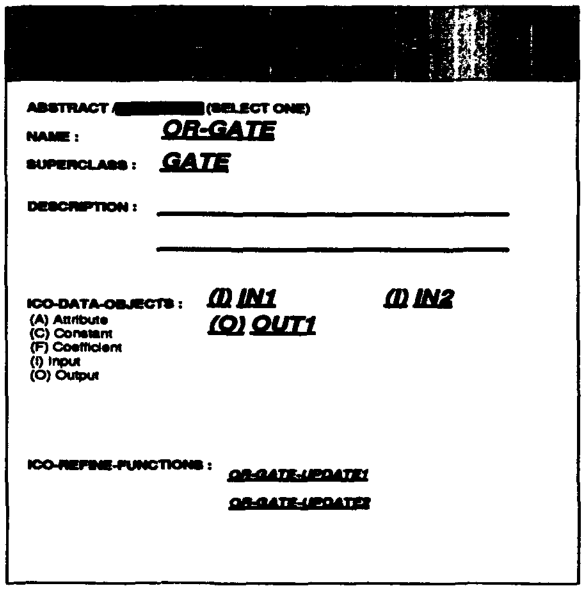
DESCRIPTION :

ICO-DATA-OBJECTS : (I) IN1 (I) IN2 (O) OUT1

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS : AND-GATE-UPDATE1 AND-GATE-UPDATE2

Figure A.15 Completed Data Collection Form for And-Gate OBJECT-CLASS



ABSTRACT / [illegible] (SELECT ONE)

NAME : OR-GATE

SUPERCLASS : GATE

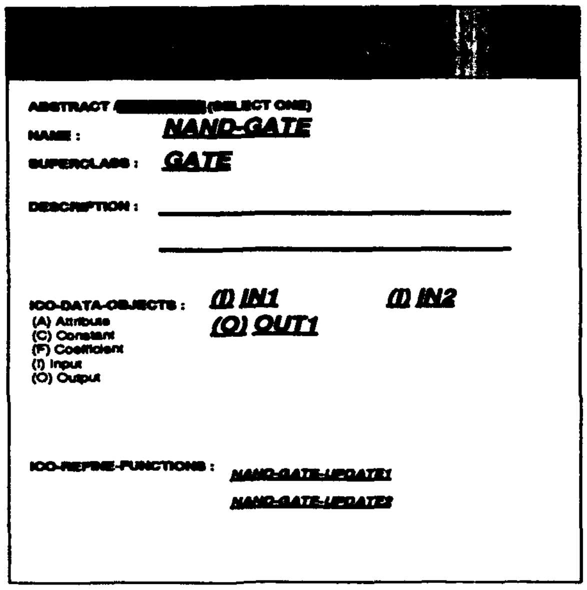
DESCRIPTION :

ICO-DATA-OBJECTS : (I) IN1 (I) IN2 (O) OUT1

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS : OR-GATE-UPDATE1 OR-GATE-UPDATE2

Figure A.16 Completed Data Collection Form for Or-Gate OBJECT-CLASS


ABSTRACT (SELECT ONE)

NAME : *NAND-GATE*

SUPERCLASS : *GATE*

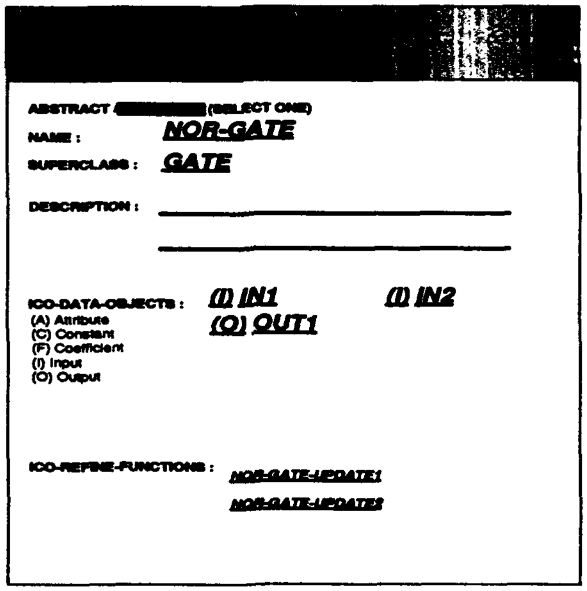
DESCRIPTION : ____________

ICO-DATA-OBJECTS : *(I) IN1* *(I) IN2* *(O) OUT1*

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS : *NAND-GATE-UPDATE1* *NAND-GATE-UPDATE2*

Figure A.17 Completed Data Collection Form for Nand-Gate OBJECT-CLASS


ABSTRACT (SELECT ONE)

NAME : *NOR-GATE*

SUPERCLASS : *GATE*

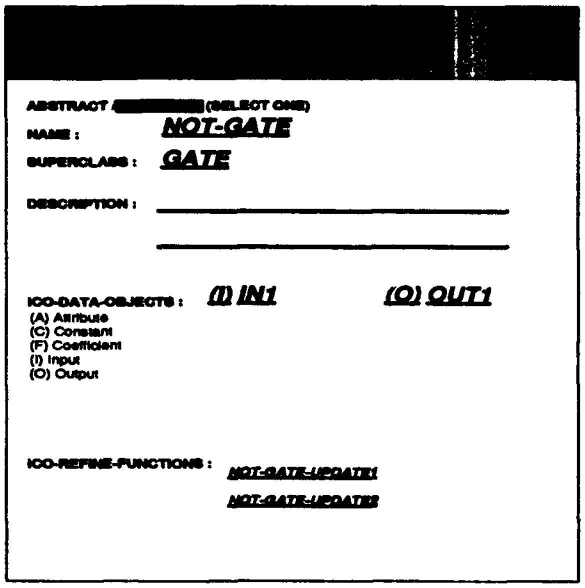
DESCRIPTION : ____________

ICO-DATA-OBJECTS : *(I) IN1* *(I) IN2* *(O) OUT1*

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS : *NOR-GATE-UPDATE1* *NOR-GATE-UPDATE2*

Figure A.18 Completed Data Collection Form for Nor-Gate OBJECT-CLASS



ABSTRACT /[illegible](SELECT ONE)

NAME : ***NOT-GATE***

SUPERCLASS : ***GATE***

DESCRIPTION : ____________________

____

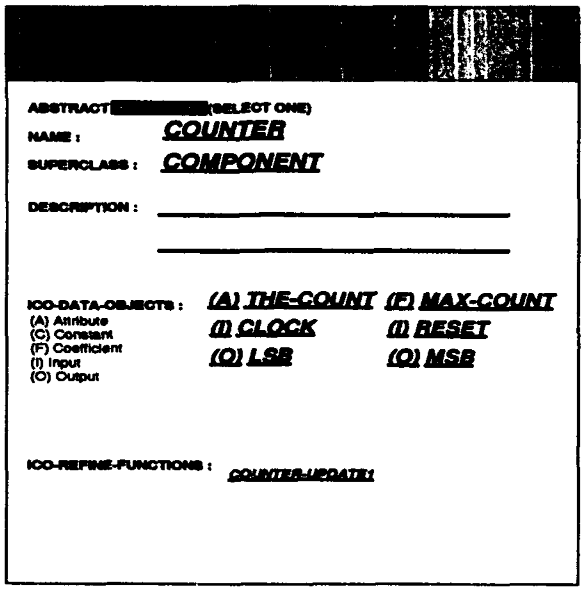
ICO-DATA-OBJECTS : ***(I) IN1*** ***(O) OUT1***

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS : ***NOT-GATE-UPDATE1***

NOT-GATE-UPDATE2

Figure A.19 Completed Data Collection Form for Not-Gate OBJECT-CLASS



ABSTRACT[illegible](SELECT ONE)

NAME : ***COUNTER***

SUPERCLASS : ***COMPONENT***

DESCRIPTION : ____________________

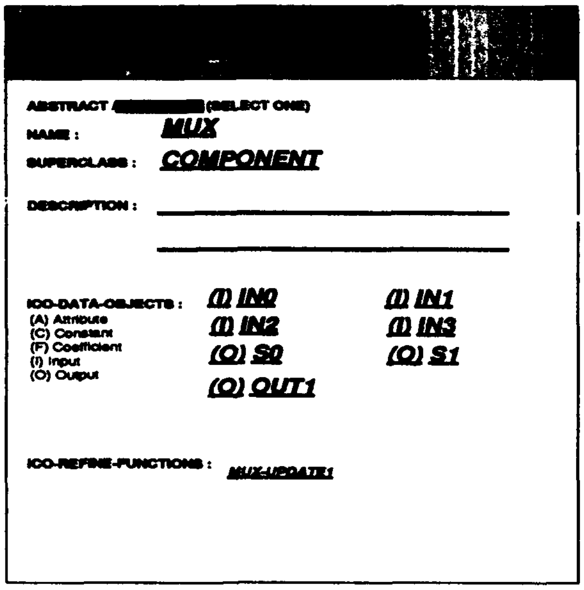
____

ICO-DATA-OBJECTS : ***(A) THE-COUNT*** ***(F) MAX-COUNT***
(I) CLOCK ***(I) RESET***
(O) LSB ***(O) MSB***

(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

ICO-REFINE-FUNCTIONS : ***COUNTER-UPDATE1***

Figure A.20 Completed Data Collection Form for Counter OBJECT-CLASS


ABSTRACT [illegible] (SELECT ONE)

NAME : ***MUX***

SUPERCLASS : ***COMPONENT***

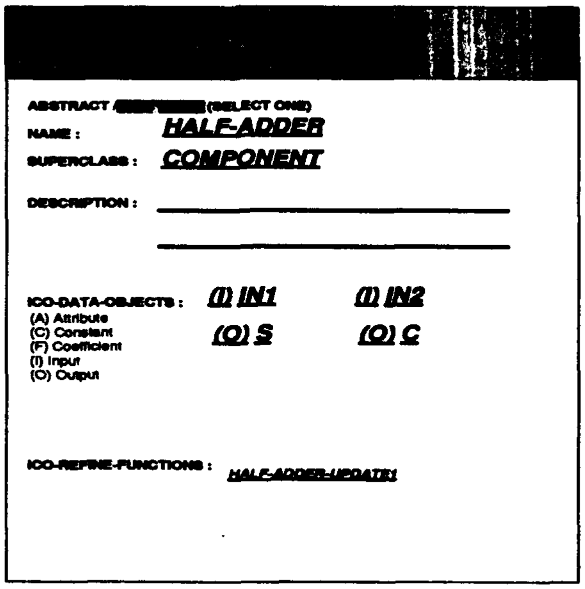
DESCRIPTION :

ICO-DATA-OBJECTS :
(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

(I) IN0	***(I) IN1***
(I) IN2	***(I) IN3***
(O) S0	***(O) S1***
(O) OUT1	

ICO-REFINE-FUNCTIONS : ***MUX-UPDATE1***

Figure A.21 Completed Data Collection Form for Multiplexer OBJECT-CLASS


ABSTRACT [illegible] (SELECT ONE)

NAME : ***HALF-ADDER***

SUPERCLASS : ***COMPONENT***

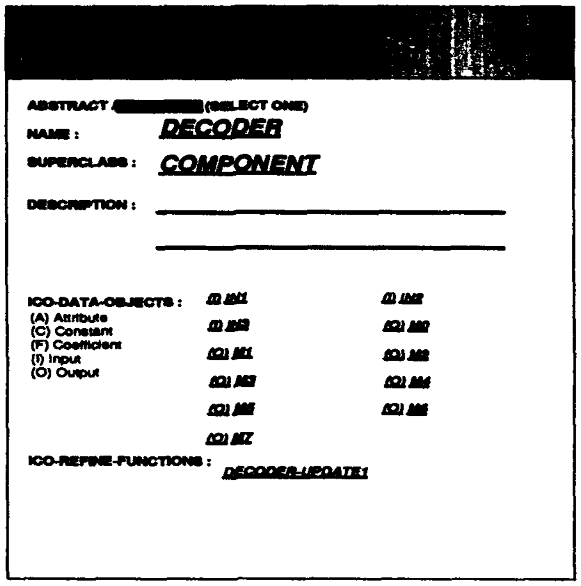
DESCRIPTION :

ICO-DATA-OBJECTS :
(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

(I) IN1	***(I) IN2***
(O) S	***(O) C***

ICO-REFINE-FUNCTIONS : ***HALF-ADDER-UPDATE1***

Figure A.22 Completed Data Collection Form for Half-Adder OBJECT-CLASS



ABSTRACT [illegible] (SELECT ONE)

NAME : DECODER

SUPERCLASS : COMPONENT

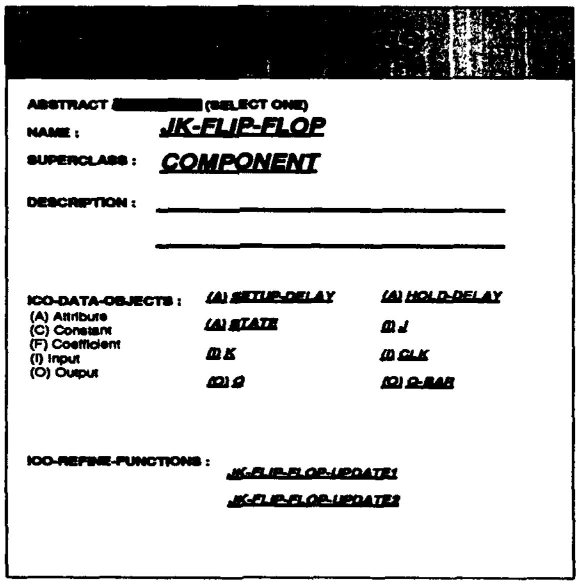
DESCRIPTION :

ICO-DATA-OBJECTS :
(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

(I) IN1	(I) IN2
(I) IN3	(O) M0
(O) M1	(O) M2
(O) M3	(O) M4
(O) M5	(O) M6
(O) M7	

ICO-REFINE-FUNCTIONS : DECODER-UPDATE1

Figure A.23 Completed Data Collection Form for Decoder OBJECT-CLASS



ABSTRACT [illegible] (SELECT ONE)

NAME : JK-FLIP-FLOP

SUPERCLASS : COMPONENT

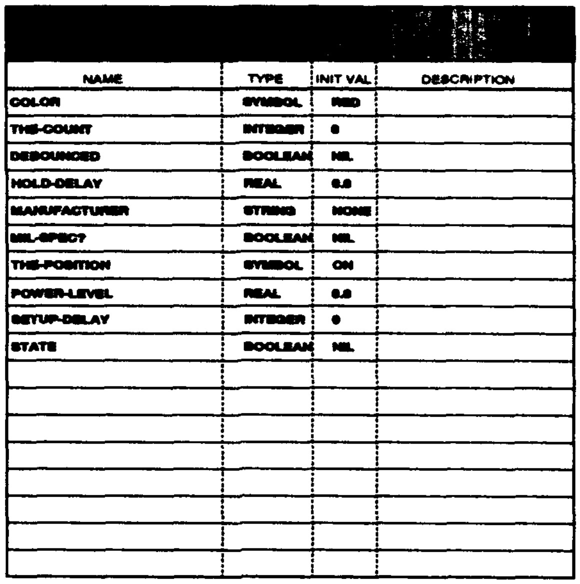
DESCRIPTION :

ICO-DATA-OBJECTS :
(A) Attribute
(C) Constant
(F) Coefficient
(I) Input
(O) Output

(A) SETUP-DELAY	(A) HOLD-DELAY
(A) STATE	(I) J
(I) K	(I) CLK
(O) Q	(O) Q-BAR

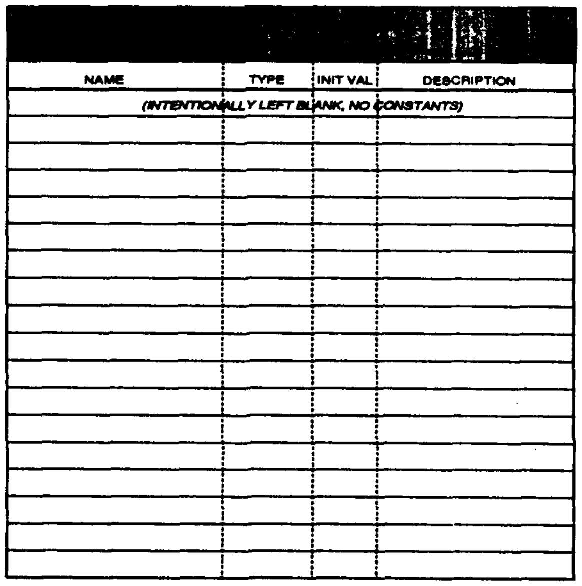
ICO-REFINE-FUNCTIONS :
JK-FLIP-FLOP-UPDATE1
JK-FLIP-FLOP-UPDATE2

Figure A.24 Completed Data Collection Form for JK-Flip-Flop OBJECT-CLASS



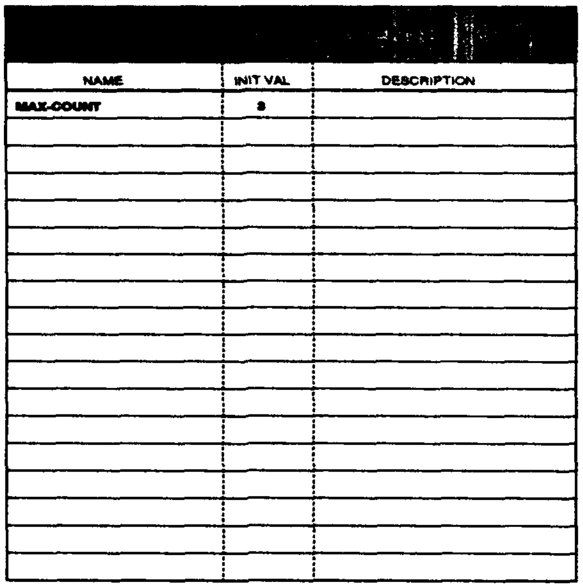
NAME	TYPE	INIT VAL	DESCRIPTION
COLOR	SYMBOL	RED	
THE-COUNT	INTEGER	0	
DEBOUNCED	BOOLEAN	NIL	
HOLD-DELAY	REAL	0.0	
MANUFACTURER	STRING	NONE	
MIL-SPEC?	BOOLEAN	NIL	
THE-POSITION	SYMBOL	ON	
POWER-LEVEL	REAL	0.0	
SETUP-DELAY	INTEGER	0	
STATE	BOOLEAN	NIL	

Figure A.25 Completed Data Collection Form for Attributes



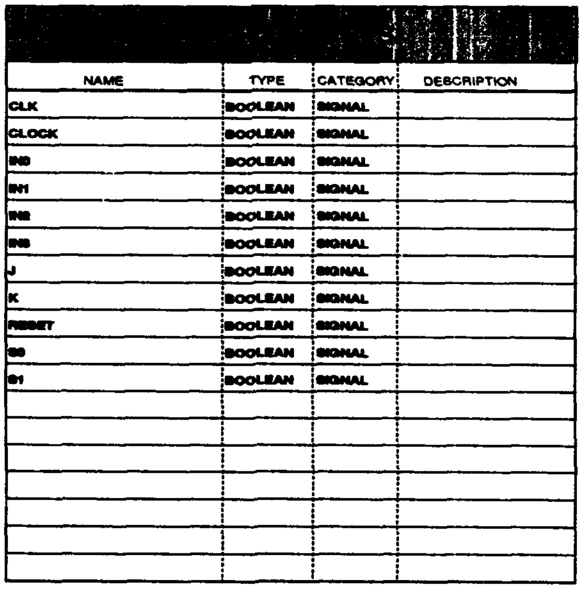
NAME	TYPE	INIT VAL	DESCRIPTION
(INTENTIONALLY LEFT BLANK, NO CONSTANTS)			

Figure A.26 Completed Data Collection Form for Constants



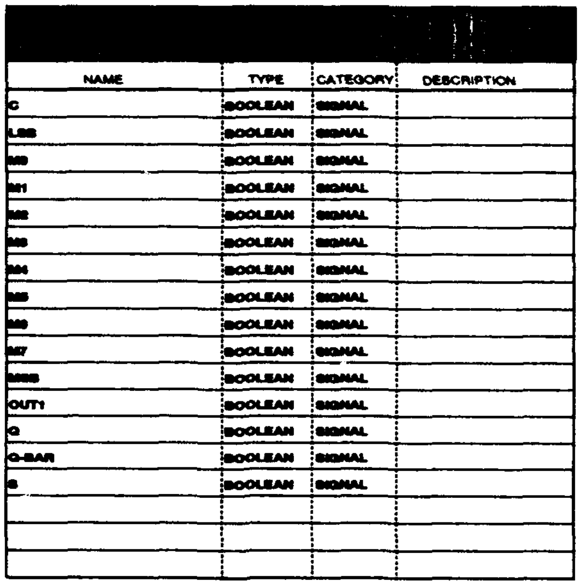
NAME	INIT VAL	DESCRIPTION
MAX-COUNT	3	

Figure A.27 Completed Data Collection Form for Coefficients



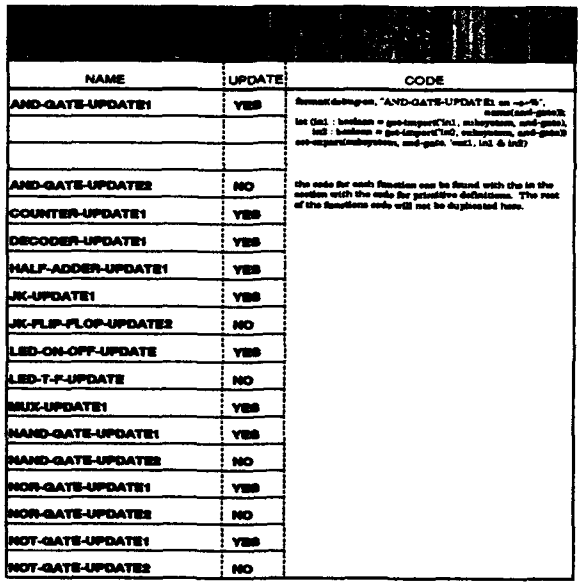
NAME	TYPE	CATEGORY	DESCRIPTION
CLK	BOOLEAN	SIGNAL	
CLOCK	BOOLEAN	SIGNAL	
IN0	BOOLEAN	SIGNAL	
IN1	BOOLEAN	SIGNAL	
IN2	BOOLEAN	SIGNAL	
IN3	BOOLEAN	SIGNAL	
J	BOOLEAN	SIGNAL	
K	BOOLEAN	SIGNAL	
RESET	BOOLEAN	SIGNAL	
S0	BOOLEAN	SIGNAL	
S1	BOOLEAN	SIGNAL	

Figure A.28 Completed Data Collection Form for OCU-Inputs



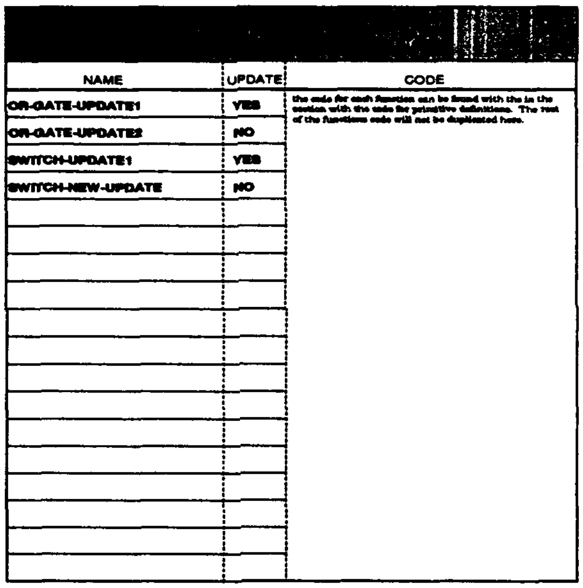
NAME	TYPE	CATEGORY	DESCRIPTION
C	BOOLEAN	SIGNAL	
LSB	BOOLEAN	SIGNAL	
M0	BOOLEAN	SIGNAL	
M1	BOOLEAN	SIGNAL	
M2	BOOLEAN	SIGNAL	
M3	BOOLEAN	SIGNAL	
M4	BOOLEAN	SIGNAL	
M5	BOOLEAN	SIGNAL	
M6	BOOLEAN	SIGNAL	
M7	BOOLEAN	SIGNAL	
MSB	BOOLEAN	SIGNAL	
OUT1	BOOLEAN	SIGNAL	
Q	BOOLEAN	SIGNAL	
Q-BAR	BOOLEAN	SIGNAL	
S	BOOLEAN	SIGNAL	

Figure A.29 Completed Data Collection Form for OCU-Outputs



NAME	UPDATE	CODE
AND-GATE-UPDATE1	YES	format(debug-on, "AND-GATE-UPDATE1 on ~s~%", name(and-gate)); let (in1 : boolean = get-import('in1, subsystem, and-gate), in2 : boolean = get-import('in2, subsystem, and-gate)) set-export(subsystem, and-gate, 'out1, in1 & in2)
AND-GATE-UPDATE2	NO	the code for each function can be found with the in the section with the code for primitive definitions. The rest of the functions code will not be duplicated here.
COUNTER-UPDATE1	YES	
DECODER-UPDATE1	YES	
HALF-ADDER-UPDATE1	YES	
JK-UPDATE1	YES	
JK-FLIP-FLOP-UPDATE2	NO	
LED-ON-OFF-UPDATE	YES	
LED-T-F-UPDATE	NO	
MUX-UPDATE1	YES	
NAND-GATE-UPDATE1	YES	
NAND-GATE-UPDATE2	NO	
NOR-GATE-UPDATE1	YES	
NOR-GATE-UPDATE2	NO	
NOT-GATE-UPDATE1	YES	
NOT-GATE-UPDATE2	NO	

Figure A.30 Completed Data Collection Form for REFINE-FUNCTION



NAME	UPDATE	CODE
OR-GATE-UPDATE1	YES	the code for each function can be found with the in the section with the code for primitive definitions. The rest of the functions code will not be duplicated here.
OR-GATE-UPDATE2	NO	
SWITCH-UPDATE1	YES	
SWITCH-NEW-UPDATE	NO	

Figure A.31 Completed Data Collection Form for REFINE-FUNCTION

Appendix B. Code Automatically Generated By ITASCA *Methods*

B.1 Database Schema Generated By ITASCA *Methods*

```
(def-class CIRCUITS
:superclasses (OCU-PRIMITIVE )
:attributes ()
:class-attributes ())

(def-class circuit-artifact
:superclasses (CIRCUITS )
:attributes (

  (manufacturer
:domain string
:init   "none specified"  )
)
:class-attributes ())

(def-class gate
:superclasses (circuit-artifact )
:attributes (

  (delay
:domain integer
:init   0  )

  (mil-spec?
:domain T
:init   nil  )

  (power-level
:domain float
:init   0.0  )
)
:class-attributes ())

(def-class nor-gate
:superclasses (gate )
:attributes (
)
:class-attributes ())

(def-class nand-gate
:superclasses (gate )
:attributes (
)
:class-attributes ())

(def-class or-gate
:superclasses (gate )
```

```
:attributes (
)
:class-attributes ())

(def-class not-gate
:superclasses (gate )
:attributes (
)
:class-attributes ())

(def-class and-gate
:superclasses (gate )
:attributes (
)
:class-attributes ())

(def-class component
:superclasses (circuit-artifact )
:attributes (

  (delay
:domain integer
:init   0  )

  (mil-spec?
:domain T
:init   nil  )

  (power-level
:domain float
:init   0.0  )
)
:class-attributes ())

(def-class decoder
:superclasses (component )
:attributes (
)
:class-attributes ())

(def-class jk-flip-flop
:superclasses (component )
:attributes (

  (set-up-delay
:domain integer
:init   0  )

  (hold-delay
:domain float
:init   0.0  )
```

```
  (state
:domain T
:init   nil  )
)
:class-attributes ())

(def-class half-adder
:superclasses (component )
:attributes (
)
:class-attributes ())

(def-class counter
:superclasses (component )
:attributes (

  (the-count
:domain integer
:init   0  )
)
:class-attributes ())

(def-class mux
:superclasses (component )
:attributes (
)
:class-attributes ())

(def-class led
:superclasses (circuit-artifact )
:attributes (

  (color
:domain symbol
:init   red  )
)
:class-attributes ())

(def-class switch
:superclasses (circuit-artifact )
:attributes (

  (delay
:domain integer
:init   0  )

  (debounced
:domain T
:init   nil  )
```

```
  (the-position
:domain symbol
:init   on  )
)
:class-attributes ())
```

B.2 REFINE *Source Code Generated By* ITASCA *Methods*

The following REFINE source code was produced by the Methods we developed in ITASCA. This source file corresponds to the digital logic circuits domain.

```
!! in-package("AVSI")

!! in-grammar('user)

"This is the CIRCUITS domain"
var CIRCUITS : object-class subtype-of Primitive-Obj

"circuit-artifact is the highest level you can add domain specific attributes"
var circuit-artifact : object-class subtype-of CIRCUITS

"superclass for all elementary circuit items"
var gate : object-class subtype-of circuit-artifact

"The NOR Gate combines its two inputs together
(after the appropriate delay) as follows

  input1 | input2 | output
  -------|--------|-------
     F   |   F    |   T
     F   |   T    |   F
     T   |   F    |   F
     T   |   T    |   F
"
var nor-gate : object-class subtype-of gate

"The NAND Gate combines its two inputs together
(after the appropriate delay) as follows

  input1 | input2 | output
  -------|--------|-------
     F   |   F    |   T
     F   |   T    |   T
     T   |   F    |   T
     T   |   T    |   F
"
var nand-gate : object-class subtype-of gate

"The OR Gate combines its two inputs together
```

```
(after the appropriate delay) as follows

  input1 | input2 | output
  -------|--------|-------
     F   |    F   |   F
     F   |    T   |   T
     T   |    F   |   T
     T   |    T   |   T
"
var or-gate : object-class subtype-of gate

"The NOT Gate takes one input and inverts it
(after the appropriate delay) as follows

  input | output
  ------|-------
     F  |    T
     T  |    F
"
var not-gate : object-class subtype-of gate

"The AND Gate combines its two inputs together
(after the appropriate delay) as follows

  input1 | input2 | output
  -------|--------|-------
     F   |    F   |   F
     F   |    T   |   F
     T   |    F   |   F
     T   |    T   |   T
"
var and-gate : object-class subtype-of gate

"superclass for all composite items in circuits domain"
var component : object-class subtype-of circuit-artifact

"description for decoder"
var decoder : object-class subtype-of component

"description for jk-flip-flop"
var jk-flip-flop : object-class subtype-of component

"description for half-adder"
var half-adder : object-class subtype-of component

"description for description for counter"
var counter : object-class subtype-of component

"description for mux"
var mux : object-class subtype-of component
```

```
"description for led"
var led : object-class subtype-of circuit-artifact

"description for switch"
var switch : object-class subtype-of circuit-artifact

var nor-gate-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in2,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var nor-gate-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'out1,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var nor-gate-coefficients : map(nor-gate, set(name-value-obj))
     computed-using
          nor-gate-coefficients(x) = {}

form MAKE-nor-gate-NAMES-UNIQUE
     unique-names-class('nor-gate, true)

"gate (abstract-class) 1"
var nor-gate-delay : map(nor-gate, integer)
              computed-using
                       nor-gate-delay(x) = 0

"gate (abstract-class) 2"
var nor-gate-mil-spec? : map(nor-gate, boolean)
              computed-using
                       nor-gate-mil-spec?(x) = nil

"gate (abstract-class) 3"
var nor-gate-power-level : map(nor-gate, real)
              computed-using
                       nor-gate-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
```

```
var nor-gate-manufacturer : map(nor-gate, string)
              computed-using
                       nor-gate-manufacturer(x) = "none specified"

function NOR-GATE-UPDATE1 (subsystem : subsystem-obj,
                              nor-gate  : NOR-GATE) =


 format(debug-on, "NOR-GATE-OBJ-UPDATE1 on ~s~%", name(nor-gate));

 let (in1 : boolean = get-import('in1, subsystem, nor-gate),
      in2 : boolean = get-import('in2, subsystem, nor-gate))

 set-export(subsystem, nor-gate, 'out1, ~(in1 or in2))


function NOR-GATE-UPDATE2 (subsystem : subsystem-obj,
                                  nor-gate  : NOR-GATE) =

 format(t, "NOR-GATE-OBJ-UPDATE2 on ~s~%", name(nor-gate))

var nor-gate-update-function : map(nor-gate, symbol)
     computed-using
          nor-gate-update-function(x) = 'NOR-GATE-UPDATE1

var decoder-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in2,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in3,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var decoder-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'm0,
          'export-category, 'signal,
          'export-type-data, 'boolean),

     set-attrs (make-object ('export-obj),
          'export-name, 'm1,
          'export-category, 'signal,
          'export-type-data, 'boolean),

     set-attrs (make-object ('export-obj),
          'export-name, 'm2,
          'export-category, 'signal,
          'export-type-data, 'boolean),

     set-attrs (make-object ('export-obj),
          'export-name, 'm3,
          'export-category, 'signal,
          'export-type-data, 'boolean),

     set-attrs (make-object ('export-obj),
          'export-name, 'm4,
          'export-category, 'signal,
          'export-type-data, 'boolean),

     set-attrs (make-object ('export-obj),
          'export-name, 'm5,
          'export-category, 'signal,
          'export-type-data, 'boolean),

     set-attrs (make-object ('export-obj),
          'export-name, 'm6,
          'export-category, 'signal,
          'export-type-data, 'boolean),

     set-attrs (make-object ('export-obj),
          'export-name, 'm7,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var decoder-coefficients : map(decoder, set(name-value-obj))
     computed-using
          decoder-coefficients(x) = {}

form MAKE-decoder-NAMES-UNIQUE
     unique-names-class('decoder, true)
```

```
"This is the delay (in nanoseconds) between the time the gate
receives it's input(s) and the time the output is available"
var decoder-delay : map(decoder, integer)
              computed-using
                       decoder-delay(x) = 0

"A true value means the gate meets exacting military specifications
A false value means the gate only meets IEEE specifications"
var decoder-mil-spec? : map(decoder, boolean)
              computed-using
                       decoder-mil-spec?(x) = nil

"component (abstract-class) 3"
var decoder-power-level : map(decoder, real)
              computed-using
                       decoder-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
var decoder-manufacturer : map(decoder, string)
              computed-using
                       decoder-manufacturer(x) = "none specified"

function DECODER-UPDATE1 (subsystem : subsystem-obj,
                              decoder   : DECODER) =

 let (x : boolean = get-import('in1, subsystem, decoder),
      y : boolean = get-import('in2, subsystem, decoder),
      z : boolean = get-import('in3, subsystem, decoder))

%  set all outputs to false to start; don't want any left-over
%     values adversely affecting output.

 set-export(subsystem, decoder, 'm0, nil);
 set-export(subsystem, decoder, 'm1, nil);
 set-export(subsystem, decoder, 'm2, nil);
 set-export(subsystem, decoder, 'm3, nil);
 set-export(subsystem, decoder, 'm4, nil);
 set-export(subsystem, decoder, 'm5, nil);
 set-export(subsystem, decoder, 'm6, nil);
 set-export(subsystem, decoder, 'm7, nil);

 if ~x and ~y and ~z then
   set-export(subsystem, decoder, 'm0, true)
 elseif ~x and ~y and z then
   set-export(subsystem, decoder, 'm1, true)
 elseif ~x and y and ~z then
   set-export(subsystem, decoder, 'm2, true)
 elseif ~x and y and z then
   set-export(subsystem, decoder, 'm3, true)
 elseif x and ~y and ~z then
   set-export(subsystem, decoder, 'm4, true)
 elseif x and ~y and z then
   set-export(subsystem, decoder, 'm5, true)
 elseif x and y and ~z then
   set-export(subsystem, decoder, 'm6, true)
 elseif x and y and z then
   set-export(subsystem, decoder, 'm7, true)

var decoder-update-function : map(decoder, symbol)
     computed-using
          decoder-update-function(x) = 'DECODER-UPDATE1

var nand-gate-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in2,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var nand-gate-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'out1,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var nand-gate-coefficients : map(nand-gate, set(name-value-obj))
     computed-using
          nand-gate-coefficients(x) = {}

form MAKE-nand-gate-NAMES-UNIQUE
     unique-names-class('nand-gate, true)

"gate (abstract-class) 1"
var nand-gate-delay : map(nand-gate, integer)
               computed-using
                         nand-gate-delay(x) = 0

"gate (abstract-class) 2"
var nand-gate-mil-spec? : map(nand-gate, boolean)
               computed-using
                         nand-gate-mil-spec?(x) = nil

"gate (abstract-class) 3"
```

```
var nand-gate-power-level : map(nand-gate, real)
              computed-using
                       nand-gate-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
var nand-gate-manufacturer : map(nand-gate, string)
              computed-using
                       nand-gate-manufacturer(x) = "none specified"

function NAND-GATE-UPDATE1 (subsystem  : subsystem-obj,
                            nand-gate  : NAND-GATE) =

 format(debug-on, "NAND-GATE-OBJ-UPDATE1 on ~s~%", name(nand-gate));

 let (in1 : boolean = get-import('in1, subsystem, nand-gate),
      in2 : boolean = get-import('in2, subsystem, nand-gate))

 set-export(subsystem, nand-gate, 'out1, ~(in1 & in2))


function NAND-GATE-UPDATE2 (subsystem  : subsystem-obj,
                            nand-gate  : NAND-GATE) =

 format(t, "NAND-GATE-OBJ-NEW-UPDATE on ~s~%", name(nand-gate))

var nand-gate-update-function : map(nand-gate, symbol)
     computed-using
          nand-gate-update-function(x) = 'NAND-GATE-UPDATE1

var or-gate-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in2,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var or-gate-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'out1,
          'export-category, 'signal,
          'export-type-data, 'boolean)}
```

```
var or-gate-coefficients : map(or-gate, set(name-value-obj))
     computed-using
          or-gate-coefficients(x) = {}

form MAKE-or-gate-NAMES-UNIQUE
     unique-names-class('or-gate, true)

"gate (abstract-class) 1"
var or-gate-delay : map(or-gate, integer)
              computed-using
                       or-gate-delay(x) = 0

"gate (abstract-class) 2"
var or-gate-mil-spec? : map(or-gate, boolean)
              computed-using
                       or-gate-mil-spec?(x) = nil

"gate (abstract-class) 3"
var or-gate-power-level : map(or-gate, real)
              computed-using
                       or-gate-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
var or-gate-manufacturer : map(or-gate, string)
              computed-using
                       or-gate-manufacturer(x) = "none specified"

function OR-GATE-UPDATE1 (subsystem : subsystem-obj,
                              or-gate   : OR-GATE) =

 format(debug-on, "OR-GATE-OBJ-UPDATE1 on ~s~%", name(or-gate));

 let (in1 : boolean = get-import('in1, subsystem, or-gate),
      in2 : boolean = get-import('in2, subsystem, or-gate))

 set-export(subsystem, or-gate, 'out1, (in1 or in2))


function OR-GATE-UPDATE2 (subsystem : subsystem-obj,
                                  or-gate   : OR-GATE) =

 format(t, "OR-GATE-OBJ-UPDATE2 on ~s~%", name(or-gate))

var or-gate-update-function : map(or-gate, symbol)
     computed-using
          or-gate-update-function(x) = 'OR-GATE-UPDATE1

var led-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var led-output-data : set(export-obj) = {
}

var led-coefficients : map(led, set(name-value-obj))
     computed-using
          led-coefficients(x) = {}

form MAKE-led-NAMES-UNIQUE
     unique-names-class('led, true)

"led  1"
var led-color : map(led, symbol)
              computed-using
                       led-color(x) = 'red

"This is the name of the company that manufactured the gate"
var led-manufacturer : map(led, string)
              computed-using
                       led-manufacturer(x) = "none specified"

function LED-ON-OFF-UPDATE (subsystem : subsystem-obj,
                                led       : LED) =

 format(debug-on, "LED-OBJ-ON-OFF-UPDATE on ~s~%", name(led));

 let (display-value : symbol = 'off)

 (if get-import('in1, subsystem, led) then
    display-value <- 'on
 );
 format(true, "LED ~s = ~s~%", name(led), display-value)


function LED-T-F-UPDATE (subsystem : subsystem-obj,
                                  led  : LED) =

 format(debug-on, "LED-OBJ-T-F-UPDATE on ~s~%", name(led));

 let (display-value : symbol = 'false)

 (if get-import('in1, subsystem, led) then
    display-value <- 'true
 );
 format(true, "LED ~s = ~s~%", name(led), display-value)

var led-update-function : map(led, symbol)
     computed-using
```

```
          led-update-function(x) = 'LED-ON-OFF-UPDATE

var jk-flip-flop-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'j,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'k,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'clk,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var jk-flip-flop-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'q,
          'export-category, 'signal,
          'export-type-data, 'boolean),


     set-attrs (make-object ('export-obj),
          'export-name, 'q-bar,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var jk-flip-flop-coefficients : map(jk-flip-flop, set(name-value-obj))
     computed-using
          jk-flip-flop-coefficients(x) = {}

form MAKE-jk-flip-flop-NAMES-UNIQUE
     unique-names-class('jk-flip-flop, true)

"jk-flip-flop 1"
var jk-flip-flop-set-up-delay : map(jk-flip-flop, integer)
               computed-using
                         jk-flip-flop-set-up-delay(x) = 0

"jk-flip-flop 2"
var jk-flip-flop-hold-delay : map(jk-flip-flop, real)
               computed-using
```

```
                  jk-flip-flop-hold-delay(x) = 0.0

"jk-flip-flop 3"
var jk-flip-flop-state : map(jk-flip-flop, boolean)
             computed-using
                  jk-flip-flop-state(x) = nil

"This is the delay (in nanoseconds) between the time the gate
receives it's input(s) and the time the output is available"
var jk-flip-flop-delay : map(jk-flip-flop, integer)
             computed-using
                  jk-flip-flop-delay(x) = 0

"A true value means the gate meets exacting military specifications
A false value means the gate only meets IEEE specifications"
var jk-flip-flop-mil-spec? : map(jk-flip-flop, boolean)
             computed-using
                  jk-flip-flop-mil-spec?(x) = nil

"component (abstract-class) 3"
var jk-flip-flop-power-level : map(jk-flip-flop, real)
             computed-using
                  jk-flip-flop-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
var jk-flip-flop-manufacturer : map(jk-flip-flop, string)
             computed-using
                  jk-flip-flop-manufacturer(x) = "none specified"

function JK-FLIP-FLOP-UPDATE1 (subsystem    : subsystem-obj,
                               jk-flip-flop : JK-FLIP-FLOP) =

 format(debug-on, "JK-FLIP-FLOP-UPDATE1 on ~s~%", name(jk-flip-flop));

 let (j   : boolean = get-import('J, subsystem, jk-flip-flop),
      k   : boolean = get-import('K, subsystem, jk-flip-flop),
      clk : boolean = get-import('Clk, subsystem, jk-flip-flop))

 (if ~j & k & clk then
    JK-FLIP-FLOP-STATE(jk-flip-flop) <- nil

 elseif j & k & clk then
    JK-FLIP-FLOP-STATE(jk-flip-flop) <- ~JK-FLIP-FLOP-STATE(jk-flip-flop)

 elseif j & ~k and clk then
    JK-FLIP-FLOP-STATE(jk-flip-flop) <- true
 );

 set-export(subsystem, jk-flip-flop, 'Q, JK-FLIP-FLOP-STATE(jk-flip-flop));
 set-export(subsystem, jk-flip-flop, 'Q-Bar, ~JK-FLIP-FLOP-STATE(jk-flip-flop))
```

```
function JK-FLIP-FLOP-UPDATE2 (subsystem    : subsystem-obj,
                               jk-flip-flop : JK-FLIP-FLOP) =

 format(t, "JK-FLIP-FLOP-UPDATE2 on ~s~%", name(jk-flip-flop))

var jk-flip-flop-update-function : map(jk-flip-flop, symbol)
     computed-using
          jk-flip-flop-update-function(x) = 'JK-FLIP-FLOP-UPDATE1

var half-adder-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in2,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var half-adder-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 's,
          'export-category, 'signal,
          'export-type-data, 'boolean),


     set-attrs (make-object ('export-obj),
          'export-name, 'c,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var half-adder-coefficients : map(half-adder, set(name-value-obj))
     computed-using
          half-adder-coefficients(x) = {}

form MAKE-half-adder-NAMES-UNIQUE
     unique-names-class('half-adder, true)

"This is the delay (in nanoseconds) between the time the gate
receives it's input(s) and the time the output is available"
var half-adder-delay : map(half-adder, integer)
               computed-using
                       half-adder-delay(x) = 0
```

```
"A true value means the gate meets exacting military specifications
A false value means the gate only meets IEEE specifications"
var half-adder-mil-spec? : map(half-adder, boolean)
              computed-using
                       half-adder-mil-spec?(x) = nil

"component (abstract-class) 3"
var half-adder-power-level : map(half-adder, real)
              computed-using
                       half-adder-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
var half-adder-manufacturer : map(half-adder, string)
              computed-using
                       half-adder-manufacturer(x) = "none specified"

function HALF-ADDER-UPDATE1 (subsystem  : subsystem-obj,
                                 half-adder : HALF-ADDER) =

 let (in1 : boolean = get-import('in1, subsystem, half-adder),
      in2 : boolean = get-import('in2, subsystem, half-adder))

 if ~in1 and ~in2 then
   set-export(subsystem, half-adder, 's, nil);
   set-export(subsystem, half-adder, 'c, nil)
 elseif in1 and ~in2 then
   set-export(subsystem, half-adder, 's, true);
   set-export(subsystem, half-adder, 'c, nil)
 elseif ~in1 and in2 then
   set-export(subsystem, half-adder, 's, true);
   set-export(subsystem, half-adder, 'c, nil)
 elseif in1 and in2 then
   set-export(subsystem, half-adder, 's, nil);
   set-export(subsystem, half-adder, 'c, true)

var half-adder-update-function : map(half-adder, symbol)
     computed-using
          half-adder-update-function(x) = 'HALF-ADDER-UPDATE1

var counter-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'clock,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'reset,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var counter-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'lsb,
          'export-category, 'signal,
          'export-type-data, 'boolean),


     set-attrs (make-object ('export-obj),
          'export-name, 'msb,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var counter-coefficients : map(counter, set(name-value-obj))
     computed-using
          counter-coefficients(x) = {
     set-attrs (make-object ('name-value-obj),
          'name-value-name, 'max-count,
          'name-value-value, 3)}

form MAKE-counter-NAMES-UNIQUE
     unique-names-class('counter, true)

"counter 1"
var counter-the-count : map(counter, integer)
               computed-using
                         counter-the-count(x) = 0

"This is the delay (in nanoseconds) between the time the gate
receives it's input(s) and the time the output is available"
var counter-delay : map(counter, integer)
               computed-using
                         counter-delay(x) = 0

"A true value means the gate meets exacting military specifications
A false value means the gate only meets IEEE specifications"
var counter-mil-spec? : map(counter, boolean)
               computed-using
                         counter-mil-spec?(x) = nil

"component (abstract-class) 3"
var counter-power-level : map(counter, real)
               computed-using
                         counter-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
var counter-manufacturer : map(counter, string)
               computed-using
```

```
                          counter-manufacturer(x) = "none specified"

function COUNTER-UPDATE1 (subsystem : subsystem-obj,
                                counter   : COUNTER) =

 format(debug-on, "COUNTER-UPDATE1 on ~s~%", name(counter));

 let (clock : boolean = get-import('clock, subsystem, counter),
      reset : boolean = get-import('reset, subsystem, counter))

 (if reset then
    COUNTER-THE-COUNT(counter) <- 0
  elseif clock then
    COUNTER-THE-COUNT(counter) <- COUNTER-THE-COUNT(counter) +1
 );
 (if COUNTER-THE-COUNT(counter) >
         get-coefficient-value(counter, 'max-count) then
    COUNTER-THE-COUNT(counter) <- 0
 );
  if COUNTER-THE-COUNT(counter) = 0 then
    set-export(subsystem, counter, 'msb, nil);
    set-export(subsystem, counter, 'lsb, nil)
  elseif COUNTER-THE-COUNT(counter) = 1 then
    set-export(subsystem, counter, 'msb, nil);
    set-export(subsystem, counter, 'lsb, true)
  elseif COUNTER-THE-COUNT(counter) = 2 then
    set-export(subsystem, counter, 'msb, true);
    set-export(subsystem, counter, 'lsb, nil)
  elseif COUNTER-THE-COUNT(counter) = 3 then
    set-export(subsystem, counter, 'msb, true);
    set-export(subsystem, counter, 'lsb, true)

var counter-update-function : map(counter, symbol)
     computed-using
          counter-update-function(x) = 'COUNTER-UPDATE1

var not-gate-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var not-gate-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'out1,
          'export-category, 'signal,
          'export-type-data, 'boolean)}
```

```
var not-gate-coefficients : map(not-gate, set(name-value-obj))
     computed-using
          not-gate-coefficients(x) = {}

form MAKE-not-gate-NAMES-UNIQUE
     unique-names-class('not-gate, true)

"gate (abstract-class) 1"
var not-gate-delay : map(not-gate, integer)
              computed-using
                       not-gate-delay(x) = 0

"gate (abstract-class) 2"
var not-gate-mil-spec? : map(not-gate, boolean)
              computed-using
                       not-gate-mil-spec?(x) = nil

"gate (abstract-class) 3"
var not-gate-power-level : map(not-gate, real)
              computed-using
                       not-gate-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
var not-gate-manufacturer : map(not-gate, string)
              computed-using
                       not-gate-manufacturer(x) = "none specified"

function NOT-GATE-UPDATE1 (subsystem : subsystem-obj,
                              not-gate  : NOT-GATE) =

 format(debug-on, "NOT-GATE-OBJ-UPDATE1 on ~s~%", name(not-gate));

 let (in1 : boolean = get-import('in1, subsystem, not-gate))

 set-export(subsystem, not-gate, 'out1, ~(in1))


function NOT-GATE-UPDATE2 (subsystem : subsystem-obj,
                                  not-gate  : NOT-GATE) =

 format(t, "NOT-GATE-OBJ-UPDATE2 on ~s~%", name(not-gate))

var not-gate-update-function : map(not-gate, symbol)
     computed-using
          not-gate-update-function(x) = 'NOT-GATE-UPDATE1

var mux-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in0,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in2,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in3,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 's1,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var mux-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'out1,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var mux-coefficients : map(mux, set(name-value-obj))
     computed-using
          mux-coefficients(x) = {}

form MAKE-mux-NAMES-UNIQUE
     unique-names-class('mux, true)

"This is the delay (in nanoseconds) between the time the gate
receives it's input(s) and the time the output is available"
var mux-delay : map(mux, integer)
          computed-using
                    mux-delay(x) = 0

"A true value means the gate meets exacting military specifications
A false value means the gate only meets IEEE specifications"
```

```
var mux-mil-spec? : map(mux, boolean)
              computed-using
                       mux-mil-spec?(x) = nil

"component (abstract-class) 3"
var mux-power-level : map(mux, real)
              computed-using
                       mux-power-level(x) = 0.0

"This is the name of the company that manufactured the gate"
var mux-manufacturer : map(mux, string)
              computed-using
                       mux-manufacturer(x) = "none specified"

function MUX-UPDATE1 (subsystem : subsystem-obj,
                         mux       : MUX) =

 let (s0 : boolean = get-import('s0, subsystem, mux),
      s1 : boolean = get-import('s1, subsystem, mux))


 if ~s0 and ~s1 then
   set-export(subsystem, mux, 'out1,
               get-import('in0, subsystem, mux))


 elseif s0 and ~s1 then
   set-export(subsystem, mux, 'out1,
               get-import('in1, subsystem, mux))


 elseif ~s0 and s1 then
   set-export(subsystem, mux, 'out1,
               get-import('in2, subsystem, mux))


 elseif s0 and s1 then
   set-export(subsystem, mux, 'out1,
               get-import('in3, subsystem, mux))

var mux-update-function : map(mux, symbol)
     computed-using
          mux-update-function(x) = 'MUX-UPDATE1

var switch-input-data : set(import-obj) = {
}

var switch-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'out1,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var switch-coefficients : map(switch, set(name-value-obj))
     computed-using
          switch-coefficients(x) = {}

form MAKE-switch-NAMES-UNIQUE
     unique-names-class('switch, true)

"switch 1"
var switch-delay : map(switch, integer)
              computed-using
                       switch-delay(x) = 0

"switch 2"
var switch-debounced : map(switch, boolean)
              computed-using
                       switch-debounced(x) = nil

"switch 3"
var switch-the-position : map(switch, symbol)
              computed-using
                       switch-the-position(x) = 'on

"This is the name of the company that manufactured the gate"
var switch-manufacturer : map(switch, string)
              computed-using
                       switch-manufacturer(x) = "none specified"

function SWITCH-UPDATE (subsystem : subsystem-obj,
                             switch    : SWITCH) =

 format(debug-on, "SWITCH-UPDATE on ~s~%", name(switch));

 let (signal : boolean = false)

 (if SWITCH-THE-POSITION(switch) = 'ON then
   signal <- true
 );

 format(t, "Switch ~S position = ~s~%", name(switch),SWITCH-THE-POSITION(switch));

 set-export(subsystem, switch, 'out1, signal)


function SWITCH-NEW-UPDATE (subsystem : subsystem-obj,
                                  switch  : SWITCH) =
 format(t, "SWITCH-NEW-UPDATE on ~s~%", name(switch))


var switch-update-function : map(switch, symbol)
     computed-using
          switch-update-function(x) = 'switch-update

var and-gate-input-data : set(import-obj) = {


     set-attrs (make-object ('import-obj),
          'import-name, 'in1,
          'import-category, 'signal,
          'import-type-data, 'boolean),


     set-attrs (make-object ('import-obj),
          'import-name, 'in2,
          'import-category, 'signal,
          'import-type-data, 'boolean)}

var and-gate-output-data : set(export-obj) = {


     set-attrs (make-object ('export-obj),
          'export-name, 'out1,
          'export-category, 'signal,
          'export-type-data, 'boolean)}

var and-gate-coefficients : map(and-gate, set(name-value-obj))
     computed-using
          and-gate-coefficients(x) = {}

form MAKE-and-gate-NAMES-UNIQUE
     unique-names-class('and-gate, true)

"gate (abstract-class) 1"
var and-gate-delay : map(and-gate, integer)
               computed-using
                         and-gate-delay(x) = 0

"gate (abstract-class) 2"
var and-gate-mil-spec? : map(and-gate, boolean)
               computed-using
                         and-gate-mil-spec?(x) = nil

"gate (abstract-class) 3"
var and-gate-power-level : map(and-gate, real)
               computed-using
                         and-gate-power-level(x) = 0.0
```

```
"This is the name of the company that manufactured the gate"
var and-gate-manufacturer : map(and-gate, string)
              computed-using
                      and-gate-manufacturer(x) = "none specified"

function AND-GATE-UPDATE1 (subsystem : subsystem-obj,
                              and-gate  : AND-GATE) =

 format(debug-on, "AND-GATE-OBJ-UPDATE on ~s~%", name(and-gate));

 let (in1 : boolean = get-import('in1, subsystem, and-gate),
      in2 : boolean = get-import('in2, subsystem, and-gate))

 set-export(subsystem, and-gate, 'out1, in1 & in2)


function AND-GATE-UPDATE2 (subsystem : subsystem-obj,
                              and-gate  : AND-GATE) =

 format(t, "AND-GATE-OBJ-UPDATE2 on ~s~%", name(and-gate))

var and-gate-update-function : map(and-gate, symbol)
     computed-using
          and-gate-update-function(x) = 'AND-GATE-UPDATE1

var nor-gate-bitmap-l   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "nor-gate-bitmap-l"))))

var nor-gate-bitmap-s   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "nor-gate-bitmap-s"))))

var decoder-bitmap-l   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "decoder-bitmap-l"))))

var decoder-bitmap-s   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "decoder-bitmap-s"))))

var nand-gate-bitmap-l   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "nand-gate-bitmap-l"))))

var nand-gate-bitmap-s   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "nand-gate-bitmap-s"))))

var or-gate-bitmap-l   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "or-gate-bitmap-l"))))

var or-gate-bitmap-s   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "or-gate-bitmap-s"))))

var led-bitmap-l   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "led-bitmap-l"))))

var led-bitmap-s   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "led-bitmap-s"))))

var jk-flip-flop-bitmap-l   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "jk-flip-flop-bitmap-l"))))

var jk-flip-flop-bitmap-s   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "jk-flip-flop-bitmap-s"))))

var half-adder-bitmap-l   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "half-adder-bitmap-l"))))

var half-adder-bitmap-s   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "half-adder-bitmap-s"))))

var counter-bitmap-l   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "counter-bitmap-l"))))

var counter-bitmap-s   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "counter-bitmap-s"))))

var not-gate-bitmap-l   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "not-gate-bitmap-l"))))

var not-gate-bitmap-s   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "not-gate-bitmap-s"))))

var mux-bitmap-l   :  any-type =
  cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
    list('equal, 'name, "mux-bitmap-l"))))

var mux-bitmap-s   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "mux-bitmap-s"))))

var switch-bitmap-l   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "switch-bitmap-l"))))

var switch-bitmap-s   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "switch-bitmap-s"))))

var and-gate-bitmap-l   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "and-gate-bitmap-l"))))

var and-gate-bitmap-s   :  any-type =
   cw::expr-to-bitmap(itasca::send('bitmap, itasca::select-any('icon-obj,
      list('equal, 'name, "and-gate-bitmap-s"))))



form build-CIRCUITS-vsl-obj
   set-attrs(make-object('viz-spec-obj),
      'name,  'CIRCUITS,
      're::zl-documentation,
"This is the CIRCUITS domain",
      'class-specs, [

set-attrs (make-object ('class-spec-obj),
   'class-name, 'nor-gate,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'nor-gate-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'nor-gate-bitmap-s)],
   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'delay,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'mil-spec?,
      'edit-attr-type, 'boolean),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'power-level,
      'edit-attr-type, 'real),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),


set-attrs (make-object ('class-spec-obj),
   'class-name, 'decoder,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'decoder-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'decoder-bitmap-s)],

   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
         'edit-attr-name, 'delay,
         'edit-attr-type, 'integer),

      set-attrs (make-object ('Edit-Attr-Obj),
         'edit-attr-name, 'mil-spec?,
         'edit-attr-type, 'boolean),

      set-attrs (make-object ('Edit-Attr-Obj),
         'edit-attr-name, 'power-level,
         'edit-attr-type, 'real),

      set-attrs (make-object ('Edit-Attr-Obj),
         'edit-attr-name, 'manufacturer,
         'edit-attr-type, 'string),

      set-attrs (make-object ('Edit-Attr-Obj),
         'edit-attr-name, 'name,
         'edit-attr-type, 'symbol)]),


   set-attrs (make-object ('class-spec-obj),
      'class-name, 'nand-gate,
      'Icon-Attributes, [

      set-attrs (make-object ('Icon-Attr-Obj),
         'icon-attr-name, 'clip-icon-label?,
         'icon-attr-val, false),

      set-attrs (make-object ('Icon-Attr-Obj),
         'icon-attr-name, 'label,
         'icon-attr-val, 'class-and-name),

      set-attrs (make-object ('Icon-Attr-Obj),
         'icon-attr-name, 'border-thickness,
         'icon-attr-val, 0),

      set-attrs (make-object ('Icon-Attr-Obj),
         'icon-attr-name, 'bitmap4icon-l,
         'icon-attr-val, 'nand-gate-bitmap-l),

      set-attrs (make-object ('Icon-Attr-Obj),
         'icon-attr-name, 'bitmap4icon-s,
         'icon-attr-val, 'nand-gate-bitmap-s)],

      'Edit-Attributes, [

      set-attrs (make-object ('Edit-Attr-Obj),
         'edit-attr-name, 'delay,
         'edit-attr-type, 'integer),

      set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'mil-spec?,
      'edit-attr-type, 'boolean),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'power-level,
      'edit-attr-type, 'real),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),


set-attrs (make-object ('class-spec-obj),
   'class-name, 'or-gate,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'or-gate-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'or-gate-bitmap-s)],

   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'delay,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'mil-spec?,
      'edit-attr-type, 'boolean),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'power-level,
      'edit-attr-type, 'real),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),


set-att   (make-object ('class-spec-obj),
   'class-name, 'led,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'led-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'led-bitmap-s)],

   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'color,
      'edit-attr-type, 'symbol),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),
set-attrs (make-object ('class-spec-obj),
   'class-name, 'jk-flip-flop,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'jk-flip-flop-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'jk-flip-flop-bitmap-s)],

   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'set-up-delay,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'hold-delay,
      'edit-attr-type, 'real),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'delay,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'mil-spec?,
      'edit-attr-type, 'boolean),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'power-level,
      'edit-attr-type, 'real),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),


set-attrs (make-object ('class-spec-obj),
   'class-name, 'half-adder,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'half-adder-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'half-adder-bitmap-s)],

   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'delay,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'mil-spec?,
      'edit-attr-type, 'boolean),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'power-level,
      'edit-attr-type, 'real),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),
set-attrs (make-object ('class-spec-obj),
   'class-name, 'counter,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'counter-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'counter-bitmap-s)],

   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'the-count,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'delay,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'mil-spec?,
      'edit-attr-type, 'boolean),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'power-level,
      'edit-attr-type, 'real),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),
set-attrs (make-object ('class-spec-obj),
   'class-name, 'not-gate,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'not-gate-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'not-gate-bitmap-s)],

   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'delay,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'mil-spec?,
      'edit-attr-type, 'boolean),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'power-level,
      'edit-attr-type, 'real),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),


set-attrs (make-object ('class-spec-obj),
   'class-name, 'mux,
   'Icon-Attributes, [
    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'clip-icon-label?,
       'icon-attr-val, false),

    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'label,
       'icon-attr-val, 'class-and-name),

    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'border-thickness,
       'icon-attr-val, 0),

    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'bitmap4icon-l,
       'icon-attr-val, 'mux-bitmap-l),

    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'bitmap4icon-s,
       'icon-attr-val, 'mux-bitmap-s)],

    'Edit-Attributes, [

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'delay,
       'edit-attr-type, 'integer),

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'mil-spec?,
       'edit-attr-type, 'boolean),

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'power-level,
       'edit-attr-type, 'real),

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'manufacturer,
       'edit-attr-type, 'string),

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'name,
       'edit-attr-type, 'symbol)]),


set-attrs (make-object ('class-spec-obj),
    'class-name, 'switch,
    'Icon-Attributes, [

    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'clip-icon-label?,
       'icon-attr-val, false),
   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'border-thickness,
      'icon-attr-val, 0),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-l,
      'icon-attr-val, 'switch-bitmap-l),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'bitmap4icon-s,
      'icon-attr-val, 'switch-bitmap-s)],

   'Edit-Attributes, [

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'delay,
      'edit-attr-type, 'integer),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'debounced,
      'edit-attr-type, 'boolean),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'the-position,
      'edit-attr-type, 'symbol),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'manufacturer,
      'edit-attr-type, 'string),

   set-attrs (make-object ('Edit-Attr-Obj),
      'edit-attr-name, 'name,
      'edit-attr-type, 'symbol)]),


set-attrs (make-object ('class-spec-obj),
   'class-name, 'and-gate,
   'Icon-Attributes, [

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'clip-icon-label?,
      'icon-attr-val, false),

   set-attrs (make-object ('Icon-Attr-Obj),
      'icon-attr-name, 'label,
      'icon-attr-val, 'class-and-name),
    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'border-thickness,
       'icon-attr-val, 0),

    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'bitmap4icon-l,
       'icon-attr-val, 'and-gate-bitmap-l),

    set-attrs (make-object ('Icon-Attr-Obj),
       'icon-attr-name, 'bitmap4icon-s,
       'icon-attr-val, 'and-gate-bitmap-s)],

    'Edit-Attributes, [

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'delay,
       'edit-attr-type, 'integer),

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'mil-spec?,
       'edit-attr-type, 'boolean),

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'power-level,
       'edit-attr-type, 'real),

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'manufacturer,
       'edit-attr-type, 'string),

    set-attrs (make-object ('Edit-Attr-Obj),
       'edit-attr-name, 'name,
       'edit-attr-type, 'symbol)])])

!! in-grammar ('syntax)

grammar  CIRCUITS

  no-patterns

  inherits-from OCU

start-classes Spec-Obj, subsystem-obj, incomplete-obj, Generic-obj, Application-obj,
nor-gate,
decoder,
nand-gate,
or-gate,
led,
jk-flip-flop,
half-adder,
counter,
not-gate,
mux,
switch,
and-gate, Expression


file-classes Spec-Obj, subsystem-obj, incomplete-obj, Generic-obj, Application-obj,
nor-gate,
decoder,
nand-gate,
or-gate,
led,
jk-flip-flop,
half-adder,
counter,
not-gate,
mux,
switch,
and-gate


productions


nor-gate           ::=    ["nor-gate" name
                           {["delay:" nor-gate-delay]
                            [(["mil-spec" !! nor-gate-mil-spec?] |
                              ["not mil-spec" ~!! nor-gate-mil-spec?])]
                            ["manufacturer:" nor-gate-manufacturer]
                            ["power level:" nor-gate-power-level] } ]
                                      builds  nor-gate,

decoder           ::=    ["decoder" name
                          {["delay:" decoder-delay]
                            [(["mil-spec" !! decoder-mil-spec?] |
                              ["not mil-spec" ~!! decoder-mil-spec?])]
                            ["manufacturer:" decoder-manufacturer]
                            ["power level:" decoder-power-level] } ]
                                      builds  decoder,

nand-gate           ::=    ["nand-gate" name
                          {["delay:" nand-gate-delay]
                            [(["is mil-spec" !! nand-gate-mil-spec?] |
                              ["not mil-spec" ~!! nand-gate-mil-spec?])]
                            ["manufacturer:" nand-gate-manufacturer]
                            ["power level:" nand-gate-power-level] } ]
                                      builds  nand-gate,
or-gate          ::=    ["or-gate" name
                          {["delay:" or-gate-delay]
                           [(["is mil-spec" !! or-gate-mil-spec?] |
                             ["not mil-spec" ~!! or-gate-mil-spec?])]
                           ["manufacturer:" or-gate-manufacturer]
                           ["power level:" or-gate-power-level] } ]
                                   builds  or-gate,

led          ::=    ["led" name
                          {["manufacturer:" led-manufacturer]
                           ["color:" led-color] } ]
                                   builds  led,

jk-flip-flop          ::=    ["jk-flip-flop" name
                          {["delay:" jk-flip-flop-delay]
                           [(["is mil-spec" !! jk-flip-flop-mil-spec?] |
                             ["not mil-spec" ~!! jk-flip-flop-mil-spec?])]
                           ["manufacturer:" jk-flip-flop-manufacturer]
                           ["power level:" jk-flip-flop-power-level]
                           ["set-up delay:" jk-flip-flop-set-up-delay]
                           ["hold delay:" jk-flip-flop-hold-delay]
                           [(["state on" !! jk-flip-flop-state]|
                             ["state off" ~!!jk-flip-flop-state]) ] } ]
                                   builds  jk-flip-flop,

half-adder          ::=    ["half-adder" name
                          {["delay:" half-adder-delay]
                           [(["mil-spec" !! half-adder-mil-spec?] |
                             ["not mil-spec" ~!! half-adder-mil-spec?])]
                           ["manufacturer:" half-adder-manufacturer]
                           ["power level:" half-adder-power-level] } ]
                                   builds  half-adder,

counter          ::=    ["counter" name
                          {["delay:" counter-delay]
                           ["count:" counter-the-count]
                           [(["mil-spec" !! counter-mil-spec?] |
                             ["not mil-spec" ~!! counter-mil-spec?])]
                           ["manufacturer:" counter-manufacturer]
                           ["power level:" counter-power-level] } ]
                                   builds  counter,

not-gate          ::=    ["not-gate" name
                          {["delay:" not-gate-delay]
                           [(["is mil-spec" !! not-gate-mil-spec?] |
                             ["not mil-spec" ~!! not-gate-mil-spec?])]
                           ["manufacturer:" not-gate-manufacturer]
                           ["power level:" not-gate-power-level] } ]
                                   builds  not-gate,

mux          ::=    ["mux" name
```

```
                         {["delay:" mux-delay]
                          [(["mil-spec" !! mux-mil-spec?] |
                            ["not mil-spec" ~!! mux-mil-spec?])]
                          ["manufacturer:" mux-manufacturer]
                          ["power level:" mux-power-level] } ]
                                   builds  mux,

switch          ::=    ["switch" name
                         {["delay:" switch-delay]
                          [(["is debounced" !! switch-debounced] |
                            ["not debounced" ~!! switch-debounced])]
                          ["manufacturer:" switch-manufacturer]
                          ["position:" switch-the-position] } ]
                                   builds  switch,

and-gate          ::=    ["and-gate" name
                         {["delay:" and-gate-delay]
                          [(["is mil-spec" !! and-gate-mil-spec?] |
                            ["not mil-spec" ~!! and-gate-mil-spec?])]
                          ["manufacturer:" and-gate-manufacturer]
                          ["power level:" and-gate-power-level] } ]
                                   builds  and-gate

  symbol-start-chars
     "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ./*"

  symbol-continue-chars
     "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-0123456789./*?"

  comments "%" matching "
"

 precedence

    for expression brackets "(" matching ")"
      (same-level "and", "or" associativity left),
      (same-level "<", "<=", "=", ">=", ">", "/=" associativity none),
      (same-level "+","-"  associativity left),
      (same-level "*", "/", "mod"  associativity left),
      (same-level "not"  associativity none),
      (same-level "abs"  associativity none),
      (same-level "**" associativity right)

  end
```

Appendix C. Technology Base Applications from Logic Circuits Domain

This appendix contains the file-based versions of the applications composed for the CIRCUITS Domain.

This is a simple test application that uses 2 switches, 2 "and" gates, and 2 LEDs.

C.1 TEST Application

```
application definition TEST
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
application TEST is controls: SUB update procedure:
  update SUB
subsystem SUB is controls: L2, L1, A2, A1, S2, S1
  imports:
    IN2 SIGNAL BOOLEAN A1 ( OUT1 SUB A2
      ) import-path: ( SUB ) import-owner: SUB
      IN1 SIGNAL BOOLEAN A1 ( OUT1 SUB S1
        ) import-path: ( SUB ) import-owner: SUB
      IN2 SIGNAL BOOLEAN A2 ( OUT1 SUB S2
        ) import-path: ( SUB ) import-owner: SUB
      IN1 SIGNAL BOOLEAN A2 ( OUT1 SUB A1
        ) import-path: ( SUB ) import-owner: SUB
      IN1 SIGNAL BOOLEAN L1 ( OUT1 SUB A1
        ) import-path: ( SUB ) import-owner: SUB
      IN1 SIGNAL BOOLEAN L2 ( OUT1 SUB A2
        ) import-path: ( SUB ) import-owner: SUB
  exports:
    OUT1 SIGNAL BOOLEAN S1
      export-path: ( SUB ) export-owner: SUB
      OUT1 SIGNAL BOOLEAN S2
        export-path: ( SUB ) export-owner: SUB
      OUT1 SIGNAL BOOLEAN A1
        export-path: ( SUB ) export-owner: SUB
      OUT1 SIGNAL BOOLEAN A2
        export-path: ( SUB ) export-owner: SUB
  initialize procedure: update procedure:
  update S1 update S2 update A1 update A2 update L1 update L2
switch S1 delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 120 comp-y: 100
switch S2 delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 120 comp-y: 200
and-gate A1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 270 comp-y: 110
and-gate A2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 272 comp-y: 234
led L1 manufacturer: " " color: RED comp-x: 434 comp-y: 114
led L2 manufacturer: " " color: RED comp-x: 434 comp-y: 241
```

C.2 ADDER Application

This application is a full adder. It is built using 2 subsystems that are half adders, built out of "and", "or", and "not" gates.

```
application definition ADDER
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
application ADDER is controls: ADDER1 update procedure:
  update ADDER1
and-gate HA1-AND1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 118 comp-y: 85
and-gate HA1-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 117 comp-y: 194
and-gate HA1-AND3 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 118 comp-y: 298
or-gate HA1-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 231 comp-y: 151
and-gate HA2-AND1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 117 comp-y: 82
and-gate HA2-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 119 comp-y: 198
and-gate HA2-AND3 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 117 comp-y: 298
or-gate HA2-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 231 comp-y: 146
switch SWITCH-1ADDEND delay: 0 not debounced
  manufacturer: " " position: OFF comp-x: 123 comp-y: 296
switch SWITCH-2ADDEND delay: 0 not debounced
  manufacturer: " " position: ON comp-x: 120 comp-y: 198
switch SWITCH-3ADDEND delay: 0 not debounced
  manufacturer: " " position: OFF comp-x: 240 comp-y: 400
led SUM-OUTPUT manufacturer: " " color: RED comp-x: 361
  comp-y: 94
led CARRY-OUTPUT manufacturer: " " color: RED comp-x: 242
  comp-y: 102
not-gate NOT-FIRSTX delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 300
not-gate NOT-FIRSTY delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 200
not-gate NOT-SECONDX delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 117 comp-y: 400
not-gate NOT-SECONDY delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 411 comp-y: 392
subsystem ADDER1 is controls:
  HA1, HA2, SWITCH-1ADDEND, SWITCH-2ADDEND, SWITCH-3ADDEND,
  NOT-FIRSTX, NOT-FIRSTY, NOT-SECONDX, NOT-SECONDY,
  SUM-OUTPUT, CARRY-OUTPUT, LAST-OR
  imports:
    IN1 SIGNAL BOOLEAN HA1-AND1 ( OUT1 ADDER1 SWITCH-1ADDEND
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
      IN2 SIGNAL BOOLEAN HA1-AND1 ( OUT1 ADDER1 NOT-FIRSTY
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA1-AND2 ( OUT1 ADDER1 NOT-FIRSTX
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN2 SIGNAL BOOLEAN HA1-AND2 ( OUT1 ADDER1 SWITCH-2ADDEND
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA1-AND3 ( OUT1 ADDER1 SWITCH-1ADDEND
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN2 SIGNAL BOOLEAN HA1-AND3 ( OUT1 ADDER1 SWITCH-2ADDEND
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA1-OR ( OUT1 HA1 HA1-AND1
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN2 SIGNAL BOOLEAN HA1-OR ( OUT1 HA1 HA1-AND2
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA2-AND1 ( OUT1 HA1 HA1-OR
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-AND1 ( OUT1 ADDER1 NOT-SECONDY
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-AND2 ( OUT1 ADDER1 NOT-SECONDX
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-AND2 ( OUT1 ADDER1 SWITCH-3ADDEND
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-AND3 ( OUT1 HA1 HA1-OR
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-AND3 ( OUT1 ADDER1 SWITCH-3ADDEND
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-OR ( OUT1 HA2 HA2-AND1
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-OR ( OUT1 HA2 HA2-AND2
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN LAST-OR ( OUT1 HA1 HA1-AND3
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN LAST-OR ( OUT1 HA2 HA2-AND3
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN CARRY-OUTPUT ( OUT1 ADDER1 LAST-OR
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN SUM-OUTPUT ( OUT1 HA2 HA2-OR
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN NOT-SECONDY ( OUT1 ADDER1 SWITCH-3ADDEND
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN NOT-SECONDX ( OUT1 HA1 HA1-OR
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN NOT-FIRSTY ( OUT1 ADDER1 SWITCH-2ADDEND
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN NOT-FIRSTX ( OUT1 ADDER1 SWITCH-1ADDEND
      ) import-path: ( ADDER1 ) import-owner: ADDER1
exports:
  OUT1 SIGNAL BOOLEAN HA1-AND1
    export-path: ( ADDER1 HA1 ) export-owner: HA1
    OUT1 SIGNAL BOOLEAN HA1-AND2
      export-path: ( ADDER1 HA1 ) export-owner: HA1
    OUT1 SIGNAL BOOLEAN HA1-AND3
```

```
        export-path: ( ADDER1 HA1 ) export-owner: HA1
      OUT1 SIGNAL BOOLEAN HA1-OR
        export-path: ( ADDER1 HA1 ) export-owner: HA1
      OUT1 SIGNAL BOOLEAN HA2-AND1
        export-path: ( ADDER1 HA2 ) export-owner: HA2
      OUT1 SIGNAL BOOLEAN HA2-AND2
        export-path: ( ADDER1 HA2 ) export-owner: HA2
      OUT1 SIGNAL BOOLEAN HA2-AND3
        export-path: ( ADDER1 HA2 ) export-owner: HA2
      OUT1 SIGNAL BOOLEAN HA2-OR
        export-path: ( ADDER1 HA2 ) export-owner: HA2
      OUT1 SIGNAL BOOLEAN LAST-OR
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN NOT-SECONDY
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN NOT-SECONDX
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN NOT-FIRSTY
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN NOT-FIRSTX
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN SWITCH-3ADDEND
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN SWITCH-2ADDEND
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN SWITCH-1ADDEND
        export-path: ( ADDER1 ) export-owner: ADDER1
  initialize procedure: update procedure:
  update SWITCH-1ADDEND update SWITCH-2ADDEND
    update SWITCH-3ADDEND update NOT-FIRSTX update NOT-FIRSTY
    update HA1 update NOT-SECONDX update NOT-SECONDY update HA2
    update LAST-OR update SUM-OUTPUT update CARRY-OUTPUT
subsystem HA1 is controls:
  HA1-AND1, HA1-AND2, HA1-AND3, HA1-OR imports: exports:
  initialize procedure: update procedure:
  update HA1-AND1 update HA1-AND2 update HA1-AND3
    update HA1-OR
  comp-x: 577 comp-y: 65
subsystem HA2 is controls:
  HA2-AND1, HA2-AND2, HA2-AND3, HA2-OR imports: exports:
  initialize procedure: update procedure:
  update HA2-AND1 update HA2-AND2 update HA2-AND3
    update HA2-OR
  comp-x: 452 comp-y: 240
or-gate LAST-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 120 comp-y: 100
```

C.3 DECODER1 Application

This application provides a test of a "decoder" primitive by providing 3 input switches and 8 output LEDs.

```
application definition DECODER1
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
switch X-IN delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 92 comp-y: 400
switch Y-IN delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 99 comp-y: 576
switch Z-IN delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 93 comp-y: 218
led M0 manufacturer: " " color: RED comp-x: 628 comp-y: 415
led M1 manufacturer: " " color: RED comp-x: 630 comp-y: 495
led M2 manufacturer: " " color: RED comp-x: 632 comp-y: 575
led M3 manufacturer: " " color: RED comp-x: 632 comp-y: 656
led M4 manufacturer: " " color: RED comp-x: 626 comp-y: 335
led M5 manufacturer: " " color: RED comp-x: 623 comp-y: 256
led M6 manufacturer: " " color: RED comp-x: 626 comp-y: 178
led M7 manufacturer: " " color: RED comp-x: 629 comp-y: 99
decoder A-DECODER delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 353 comp-y: 388
subsystem DO-DECODE is controls:
  X-IN, Y-IN, Z-IN, M0, M1, M2, M3, M4,
  M5, M6, M7, A-DECODER
  imports:
    IN3 SIGNAL BOOLEAN A-DECODER ( OUT1 DO-DECODE Z-IN
      ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN2 SIGNAL BOOLEAN A-DECODER ( OUT1 DO-DECODE Y-IN
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN A-DECODER ( OUT1 DO-DECODE X-IN
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN M7 ( M7 DO-DECODE A-DECODER
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN M6 ( M6 DO-DECODE A-DECODER
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN M5 ( M5 DO-DECODE A-DECODER
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN M4 ( M4 DO-DECODE A-DECODER
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN M3 ( M3 DO-DECODE A-DECODER
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN M2 ( M2 DO-DECODE A-DECODER
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN M1 ( M1 DO-DECODE A-DECODER
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
      IN1 SIGNAL BOOLEAN M0 ( M0 DO-DECODE A-DECODER
        ) import-path: ( DO-DECODE ) import-owner: DO-DECODE
  exports:
    M7 SIGNAL BOOLEAN A-DECODER
```

```
      export-path: ( DO-DECODE ) export-owner: DO-DECODE
      M6 SIGNAL BOOLEAN A-DECODER
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      M5 SIGNAL BOOLEAN A-DECODER
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      M4 SIGNAL BOOLEAN A-DECODER
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      M3 SIGNAL BOOLEAN A-DECODER
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      M2 SIGNAL BOOLEAN A-DECODER
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      M1 SIGNAL BOOLEAN A-DECODER
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      M0 SIGNAL BOOLEAN A-DECODER
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      OUT1 SIGNAL BOOLEAN Z-IN
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      OUT1 SIGNAL BOOLEAN Y-IN
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
      OUT1 SIGNAL BOOLEAN X-IN
        export-path: ( DO-DECODE ) export-owner: DO-DECODE
  initialize procedure: update procedure:
  update X-IN update Y-IN update Z-IN update A-DECODER
    update M0 update M1 update M2 update M3 update M4 update M5
    update M6 update M7
application DECODER1 is controls: DO-DECODE
  update procedure: update DO-DECODE
```

C.4 ADD-AND-DECODE Application

This application tests the combination of an adder and a decoder. The adder is a subsystem composed of 2 half-adder subsystems, each composed out of "and" gates, "or" gates, and "not" gates. The decoder is also a subsystem, composed of "and" gates and "not" gates.

```
application definition ADD-AND-DECODE
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
application ADD-AND-DECODE is controls:
  DRIVE-DECODER1, ADDER1 update procedure:
  update ADDER1 update DRIVE-DECODER1
switch SWITCH-NULL delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 360 comp-y: 100
subsystem DRIVE-DECODER1 is controls:
  SWITCH-NULL, M0, M1, M2, M3, M4, M5, M6,
  M7, DECODER1
  imports:
    IN2 SIGNAL BOOLEAN AND-M7-B
      ( OUT1 DRIVE-DECODER1 SWITCH-NULL
      ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
          DECODER1
```

```
IN1 SIGNAL BOOLEAN AND-M7-B ( OUT1 DECODER1 AND-M7-A
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN2 SIGNAL BOOLEAN AND-M7-A ( OUT1 ADDER1 LAST-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN1 SIGNAL BOOLEAN AND-M7-A ( OUT1 HA2 HA2-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN2 SIGNAL BOOLEAN AND-M6-B
  ( OUT1 DRIVE-DECODER1 SWITCH-NULL
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN1 SIGNAL BOOLEAN AND-M6-B ( OUT1 DECODER1 AND-M6-A
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN2 SIGNAL BOOLEAN AND-M6-A ( OUT1 ADDER1 LAST-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN1 SIGNAL BOOLEAN AND-M6-A ( OUT1 DECODER1 NOT-Z
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN2 SIGNAL BOOLEAN AND-M5-B
  ( OUT1 DRIVE-DECODER1 SWITCH-NULL
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN1 SIGNAL BOOLEAN AND-M5-B ( OUT1 DECODER1 AND-M5-A
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN2 SIGNAL BOOLEAN AND-M5-A ( OUT1 DECODER1 NOT-Y
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN1 SIGNAL BOOLEAN AND-M5-A ( OUT1 HA2 HA2-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN2 SIGNAL BOOLEAN AND-M4-B
  ( OUT1 DRIVE-DECODER1 SWITCH-NULL
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN1 SIGNAL BOOLEAN AND-M4-B ( OUT1 DECODER1 AND-M4-A
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN2 SIGNAL BOOLEAN AND-M4-A ( OUT1 DECODER1 NOT-Y
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN1 SIGNAL BOOLEAN AND-M4-A ( OUT1 DECODER1 NOT-Z
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
IN2 SIGNAL BOOLEAN AND-M3-B ( OUT1 DECODER1 NOT-X
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
      DECODER1
```

```
IN1 SIGNAL BOOLEAN AND-M3-B ( OUT1 DECODER1 AND-M3-A
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN2 SIGNAL BOOLEAN AND-M3-A ( OUT1 ADDER1 LAST-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN AND-M3-A ( OUT1 HA2 HA2-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN2 SIGNAL BOOLEAN AND-M2-B ( OUT1 DECODER1 NOT-X
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN AND-M2-B ( OUT1 DECODER1 AND-M2-A
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN2 SIGNAL BOOLEAN AND-M2-A ( OUT1 ADDER1 LAST-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN AND-M2-A ( OUT1 DECODER1 NOT-Z
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN2 SIGNAL BOOLEAN AND-M1-B ( OUT1 DECODER1 NOT-X
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN AND-M1-B ( OUT1 DECODER1 AND-M1-A
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN2 SIGNAL BOOLEAN AND-M1-A ( OUT1 DECODER1 NOT-Y
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN AND-M1-A ( OUT1 HA2 HA2-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN2 SIGNAL BOOLEAN AND-M0-B ( OUT1 DECODER1 NOT-X
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN AND-M0-B ( OUT1 DECODER1 AND-M0-A
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN2 SIGNAL BOOLEAN AND-M0-A ( OUT1 DECODER1 NOT-Y
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN AND-M0-A ( OUT1 DECODER1 NOT-Z
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN NOT-Z ( OUT1 HA2 HA2-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
IN1 SIGNAL BOOLEAN NOT-Y ( OUT1 ADDER1 LAST-OR
  ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
     DECODER1
```

```
      IN1 SIGNAL BOOLEAN NOT-X ( OUT1 DRIVE-DECODER1 SWITCH-NULL
        ) import-path: ( DRIVE-DECODER1 DECODER1 ) import-owner:
          DECODER1
      IN1 SIGNAL BOOLEAN M7 ( OUT1 DECODER1 AND-M7-B
        ) import-path: ( DRIVE-DECODER1 ) import-owner:
          DRIVE-DECODER1
      IN1 SIGNAL BOOLEAN M6 ( OUT1 DECODER1 AND-M6-B
        ) import-path: ( DRIVE-DECODER1 ) import-owner:
          DRIVE-DECODER1
      IN1 SIGNAL BOOLEAN M5 ( OUT1 DECODER1 AND-M5-B
        ) import-path: ( DRIVE-DECODER1 ) import-owner:
          DRIVE-DECODER1
      IN1 SIGNAL BOOLEAN M4 ( OUT1 DECODER1 AND-M4-B
        ) import-path: ( DRIVE-DECODER1 ) import-owner:
          DRIVE-DECODER1
      IN1 SIGNAL BOOLEAN M3 ( OUT1 DECODER1 AND-M3-B
        ) import-path: ( DRIVE-DECODER1 ) import-owner:
          DRIVE-DECODER1
      IN1 SIGNAL BOOLEAN M2 ( OUT1 DECODER1 AND-M2-B
        ) import-path: ( DRIVE-DECODER1 ) import-owner:
          DRIVE-DECODER1
      IN1 SIGNAL BOOLEAN M1 ( OUT1 DECODER1 AND-M1-B
        ) import-path: ( DRIVE-DECODER1 ) import-owner:
          DRIVE-DECODER1
      IN1 SIGNAL BOOLEAN M0 ( OUT1 DECODER1 AND-M0-B
        ) import-path: ( DRIVE-DECODER1 ) import-owner:
          DRIVE-DECODER1
  exports:
    OUT1 SIGNAL BOOLEAN AND-M7-B
      export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
        DECODER1
      OUT1 SIGNAL BOOLEAN AND-M7-A
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M6-B
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M6-A
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M5-B
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M5-A
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M4-B
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M4-A
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
```

```
        DECODER1
      OUT1 SIGNAL BOOLEAN AND-M3-B
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M3-A
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M2-B
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M2-A
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M1-B
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M1-A
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M0-B
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN AND-M0-A
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN NOT-Z
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN NOT-Y
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN NOT-X
        export-path: ( DRIVE-DECODER1 DECODER1 ) export-owner:
          DECODER1
      OUT1 SIGNAL BOOLEAN SWITCH-NULL
        export-path: ( DRIVE-DECODER1 ) export-owner: DRIVE-DECODER1
  initialize procedure: update procedure:
  update SWITCH-NULL update DECODER1 update M0 update M1
    update M2 update M3 update M4 update M5 update M6 update M7
and-gate AND-M0-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 597 comp-y: 753
and-gate AND-M0-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 921 comp-y: 762
and-gate AND-M1-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 599 comp-y: 664
and-gate AND-M1-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 924 comp-y: 665
and-gate AND-M2-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 602 comp-y: 574
and-gate AND-M2-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 925 comp-y: 575
```

```
and-gate AND-M3-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 601 comp-y: 484
and-gate AND-M3-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 926 comp-y: 486
and-gate AND-M4-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 604 comp-y: 391
and-gate AND-M4-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 928 comp-y: 401
and-gate AND-M5-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 608 comp-y: 299
and-gate AND-M5-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 929 comp-y: 314
and-gate AND-M6-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 615 comp-y: 205
and-gate AND-M6-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 928 comp-y: 217
and-gate AND-M7-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 616 comp-y: 93
and-gate AND-M7-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 923 comp-y: 106
led M0 manufacturer: " " color: RED comp-x: 240 comp-y: 400
led M1 manufacturer: " " color: RED comp-x: 240 comp-y: 300
led M2 manufacturer: " " color: RED comp-x: 240 comp-y: 200
led M3 manufacturer: " " color: RED comp-x: 240 comp-y: 100
led M4 manufacturer: " " color: RED comp-x: 120 comp-y: 400
led M5 manufacturer: " " color: RED comp-x: 120 comp-y: 300
led M6 manufacturer: " " color: RED comp-x: 120 comp-y: 200
led M7 manufacturer: " " color: RED comp-x: 120 comp-y: 100
not-gate NOT-X delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 733 comp-y: 601
not-gate NOT-Y delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 208 comp-y: 505
not-gate NOT-Z delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 203 comp-y: 673
subsystem DECODER1 is controls:
  NOT-X, NOT-Y, NOT-Z, AND-M0-A, AND-M0-B, AND-M1-A,
  AND-M1-B, AND-M2-A, AND-M2-B, AND-M3-A, AND-M3-B, AND-M4-A,
  AND-M4-B, AND-M5-A, AND-M5-B, AND-M6-A, AND-M6-B, AND-M7-A,
  AND-M7-B
  imports: exports: initialize procedure: update procedure:
  update NOT-X update NOT-Y update NOT-Z update AND-M0-A
    update AND-M0-B update AND-M1-A update AND-M1-B
    update AND-M2-A update AND-M2-B update AND-M3-A
    update AND-M3-B update AND-M4-A update AND-M4-B
    update AND-M5-A update AND-M5-B update AND-M6-A
    update AND-M6-B update AND-M7-A update AND-M7-B
  comp-x: 446 comp-y: 259
subsystem ADDER1 is controls:
  HA1, HA2, SWITCH-1ADDEND, SWITCH-2ADDEND, SWITCH-3ADDEND,
  NOT-FIRSTX, NOT-FIRSTY, NOT-SECONDX, NOT-SECONDY,
  SUM-OUTPUT, CARRY-OUTPUT, LAST-OR
```

```
imports:
  IN1 SIGNAL BOOLEAN HA1-AND1 ( OUT1 ADDER1 SWITCH-1ADDEND
    ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN2 SIGNAL BOOLEAN HA1-AND1 ( OUT1 ADDER1 NOT-FIRSTY
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA1-AND2 ( OUT1 ADDER1 NOT-FIRSTX
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN2 SIGNAL BOOLEAN HA1-AND2 ( OUT1 ADDER1 SWITCH-2ADDEND
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA1-AND3 ( OUT1 ADDER1 SWITCH-1ADDEND
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN2 SIGNAL BOOLEAN HA1-AND3 ( OUT1 ADDER1 SWITCH-2ADDEND
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA1-OR ( OUT1 HA1 HA1-AND1
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN2 SIGNAL BOOLEAN HA1-OR ( OUT1 HA1 HA1-AND2
      ) import-path: ( ADDER1 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA2-AND1 ( OUT1 HA1 HA1-OR
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-AND1 ( OUT1 ADDER1 NOT-SECONDY
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-AND2 ( OUT1 ADDER1 NOT-SECONDX
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-AND2 ( OUT1 ADDER1 SWITCH-3ADDEND
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-AND3 ( OUT1 HA1 HA1-OR
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-AND3 ( OUT1 ADDER1 SWITCH-3ADDEND
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-OR ( OUT1 HA2 HA2-AND1
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-OR ( OUT1 HA2 HA2-AND2
      ) import-path: ( ADDER1 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN LAST-OR ( OUT1 HA1 HA1-AND3
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN LAST-OR ( OUT1 HA2 HA2-AND3
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN CARRY-OUTPUT ( OUT1 ADDER1 LAST-OR
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN SUM-OUTPUT ( OUT1 HA2 HA2-OR
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN NOT-SECONDY ( OUT1 ADDER1 SWITCH-3ADDEND
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN NOT-SECONDX ( OUT1 HA1 HA1-OR
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN NOT-FIRSTY ( OUT1 ADDER1 SWITCH-2ADDEND
      ) import-path: ( ADDER1 ) import-owner: ADDER1
    IN1 SIGNAL BOOLEAN NOT-FIRSTX ( OUT1 ADDER1 SWITCH-1ADDEND
      ) import-path: ( ADDER1 ) import-owner: ADDER1
exports:
  OUT1 SIGNAL BOOLEAN HA1-AND1
```

```
      export-path: ( ADDER1 HA1 ) export-owner: HA1
      OUT1 SIGNAL BOOLEAN HA1-AND2
        export-path: ( ADDER1 HA1 ) export-owner: HA1
      OUT1 SIGNAL BOOLEAN HA1-AND3
        export-path: ( ADDER1 HA1 ) export-owner: HA1
      OUT1 SIGNAL BOOLEAN HA1-OR
        export-path: ( ADDER1 HA1 ) export-owner: HA1
      OUT1 SIGNAL BOOLEAN HA2-AND1
        export-path: ( ADDER1 HA2 ) export-owner: HA2
      OUT1 SIGNAL BOOLEAN HA2-AND2
        export-path: ( ADDER1 HA2 ) export-owner: HA2
      OUT1 SIGNAL BOOLEAN HA2-AND3
        export-path: ( ADDER1 HA2 ) export-owner: HA2
      OUT1 SIGNAL BOOLEAN HA2-OR
        export-path: ( ADDER1 HA2 ) export-owner: HA2
      OUT1 SIGNAL BOOLEAN LAST-OR
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN NOT-SECONDY
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN NOT-SECONDX
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN NOT-FIRSTY
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN NOT-FIRSTX
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN SWITCH-3ADDEND
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN SWITCH-2ADDEND
        export-path: ( ADDER1 ) export-owner: ADDER1
      OUT1 SIGNAL BOOLEAN SWITCH-1ADDEND
        export-path: ( ADDER1 ) export-owner: ADDER1
  initialize procedure: update procedure:
  update SWITCH-1ADDEND update SWITCH-2ADDEND
    update SWITCH-3ADDEND update NOT-FIRSTX update NOT-FIRSTY
    update HA1 update NOT-SECONDX update NOT-SECONDY update HA2
    update LAST-OR update SUM-OUTPUT update CARRY-OUTPUT
and-gate HA1-AND1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 121 comp-y: 90
and-gate HA1-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 124 comp-y: 199
and-gate HA1-AND3 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 119 comp-y: 305
or-gate HA1-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 241 comp-y: 139
subsystem HA1 is controls:
  HA1-AND1, HA1-AND2, HA1-AND3, HA1-OR imports: exports:
  initialize procedure: update procedure:
  update HA1-AND1 update HA1-AND2 update HA1-AND3
    update HA1-OR
  comp-x: 449 comp-y: 334
and-gate HA2-AND1 delay: 0 not mil-spec manufacturer: " "
```

```
  power level: 0.0 comp-x: 115 comp-y: 72
and-gate HA2-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 118 comp-y: 193
and-gate HA2-AND3 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 118 comp-y: 306
or-gate HA2-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 227 comp-y: 131
subsystem HA2 is controls:
  HA2-AND1, HA2-AND2, HA2-AND3, HA2-OR imports: exports:
  initialize procedure: update procedure:
  update HA2-AND1 update HA2-AND2 update HA2-AND3
    update HA2-OR
  comp-x: 451 comp-y: 111
or-gate LAST-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 92 comp-y: 387
not-gate NOT-FIRSTX delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 245 comp-y: 108
not-gate NOT-FIRSTY delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 200
not-gate NOT-SECONDX delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 89 comp-y: 486
not-gate NOT-SECONDY delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 242 comp-y: 298
led SUM-OUTPUT manufacturer: " " color: RED comp-x: 227
  comp-y: 480
led CARRY-OUTPUT manufacturer: " " color: RED comp-x: 233
  comp-y: 389
switch SWITCH-1ADDEND delay: 0 not debounced
  manufacturer: " " position: ON comp-x: 113 comp-y: 104
switch SWITCH-2ADDEND delay: 0 not debounced
  manufacturer: " " position: ON comp-x: 105 comp-y: 200
switch SWITCH-3ADDEND delay: 0 not debounced
  manufacturer: " " position: OFF comp-x: 103 comp-y: 291
```

C.5 ADD-AND-DECODE2 Application

This application is similar to the ADD-AND-DECODE application mentioned above, except the decoder subsystem is implemented as a decoder with output (i.e., the LEDs have been moved down under the control of the decoder subsystem).

```
application definition ADD-AND-DECODE2
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
application ADD-AND-DECODE2 is controls:
  DRIVE-DECODER, ADDER3 update procedure:
  update ADDER3 update DRIVE-DECODER
switch SWITCH-NULL delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 120 comp-y: 100
subsystem DRIVE-DECODER is controls:
  SWITCH-NULL, DECODER-WITH-OUTPUT
```

```
imports:
  IN1 SIGNAL BOOLEAN M7 ( OUT1 DECODER-WITH-OUTPUT AND-M7-B
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN M6 ( OUT1 DECODER-WITH-OUTPUT AND-M6-B
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN M5 ( OUT1 DECODER-WITH-OUTPUT AND-M5-B
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN M4 ( OUT1 DECODER-WITH-OUTPUT AND-M4-B
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN M3 ( OUT1 DECODER-WITH-OUTPUT AND-M3-B
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN M2 ( OUT1 DECODER-WITH-OUTPUT AND-M2-B
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN M1 ( OUT1 DECODER-WITH-OUTPUT AND-M1-B
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN M0 ( OUT1 DECODER-WITH-OUTPUT AND-M0-B
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN2 SIGNAL BOOLEAN AND-M7-B ( OUT1 DRIVE-DECODER SWITCH-NULL
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN AND-M7-B
      ( OUT1 DECODER-WITH-OUTPUT AND-M7-A
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN2 SIGNAL BOOLEAN AND-M7-A ( OUT1 ADDER3 LAST-OR
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN AND-M7-A ( OUT1 HA2 HA2-NOT
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN2 SIGNAL BOOLEAN AND-M6-B ( OUT1 DRIVE-DECODER SWITCH-NULL
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN AND-M6-B
      ( OUT1 DECODER-WITH-OUTPUT AND-M6-A
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN2 SIGNAL BOOLEAN AND-M6-A ( OUT1 ADDER3 LAST-OR
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
    IN1 SIGNAL BOOLEAN AND-M6-A ( OUT1 DECODER-WITH-OUTPUT NOT-Z
      ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
          ) import-owner: DECODER-WITH-OUTPUT
```

```
IN2 SIGNAL BOOLEAN AND-M5-B ( OUT1 DRIVE-DECODER SWITCH-NULL
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN1 SIGNAL BOOLEAN AND-M5-B
  ( OUT1 DECODER-WITH-OUTPUT AND-M5-A
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN2 SIGNAL BOOLEAN AND-M5-A ( OUT1 DECODER-WITH-OUTPUT NOT-Y
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN1 SIGNAL BOOLEAN AND-M5-A ( OUT1 HA2 HA2-NOT
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN2 SIGNAL BOOLEAN AND-M4-B ( OUT1 DRIVE-DECODER SWITCH-NULL
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN1 SIGNAL BOOLEAN AND-M4-B
  ( OUT1 DECODER-WITH-OUTPUT AND-M4-A
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN2 SIGNAL BOOLEAN AND-M4-A ( OUT1 DECODER-WITH-OUTPUT NOT-Y
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN1 SIGNAL BOOLEAN AND-M4-A ( OUT1 DECODER-WITH-OUTPUT NOT-Z
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN2 SIGNAL BOOLEAN AND-M3-B ( OUT1 DECODER-WITH-OUTPUT NOT-X
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN1 SIGNAL BOOLEAN AND-M3-B
  ( OUT1 DECODER-WITH-OUTPUT AND-M3-A
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN2 SIGNAL BOOLEAN AND-M3-A ( OUT1 HA2 HA2-NOT
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN1 SIGNAL BOOLEAN AND-M3-A ( OUT1 ADDER3 LAST-OR
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN2 SIGNAL BOOLEAN AND-M2-B ( OUT1 DECODER-WITH-OUTPUT NOT-X
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN1 SIGNAL BOOLEAN AND-M2-B
  ( OUT1 DECODER-WITH-OUTPUT AND-M2-A
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN2 SIGNAL BOOLEAN AND-M2-A ( OUT1 ADDER3 LAST-OR
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) import-owner: DECODER-WITH-OUTPUT
IN1 SIGNAL BOOLEAN AND-M2-A ( OUT1 DECODER-WITH-OUTPUT NOT-Z
  ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
```

```
        ) import-owner: DECODER-WITH-OUTPUT
  IN2 SIGNAL BOOLEAN AND-M1-B ( OUT1 DECODER-WITH-OUTPUT NOT-X
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN1 SIGNAL BOOLEAN AND-M1-B
    ( OUT1 DECODER-WITH-OUTPUT AND-M1-A
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN2 SIGNAL BOOLEAN AND-M1-A ( OUT1 DECODER-WITH-OUTPUT NOT-Y
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN1 SIGNAL BOOLEAN AND-M1-A ( OUT1 HA2 HA2-NOT
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN2 SIGNAL BOOLEAN AND-M0-B ( OUT1 DECODER-WITH-OUTPUT NOT-X
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN1 SIGNAL BOOLEAN AND-M0-B
    ( OUT1 DECODER-WITH-OUTPUT AND-M0-A
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN2 SIGNAL BOOLEAN AND-M0-A ( OUT1 DECODER-WITH-OUTPUT NOT-Y
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN1 SIGNAL BOOLEAN AND-M0-A ( OUT1 DECODER-WITH-OUTPUT NOT-Z
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN1 SIGNAL BOOLEAN NOT-Z ( OUT1 HA2 HA2-NOT
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN1 SIGNAL BOOLEAN NOT-Y ( OUT1 ADDER3 LAST-OR
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
  IN1 SIGNAL BOOLEAN NOT-X ( OUT1 DRIVE-DECODER SWITCH-NULL
    ) import-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) import-owner: DECODER-WITH-OUTPUT
exports:
  OUT1 SIGNAL BOOLEAN AND-M7-B
    export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
      ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M7-A
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M6-B
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M6-A
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M5-B
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M5-A
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M4-B
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M4-A
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M3-B
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M3-A
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M2-B
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M2-A
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M1-B
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M1-A
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M0-B
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN AND-M0-A
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN NOT-Z
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN NOT-Y
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN NOT-X
      export-path: ( DRIVE-DECODER DECODER-WITH-OUTPUT
        ) export-owner: DECODER-WITH-OUTPUT
    OUT1 SIGNAL BOOLEAN SWITCH-NULL
      export-path: ( DRIVE-DECODER ) export-owner: DRIVE-DECODER
  initialize procedure: update procedure:
  update SWITCH-NULL update DECODER-WITH-OUTPUT
switch SWITCH-1ADDEND delay: 0 not debounced
  manufacturer: " " position: ON comp-x: 120 comp-y: 300
switch SWITCH-2ADDEND delay: 0 not debounced
  manufacturer: " " position: OFF comp-x: 118 comp-y: 402
```

```
switch SWITCH-3ADDEND delay: 0 not debounced
  manufacturer: " " position: OFF comp-x: 120 comp-y: 100
and-gate HA1-AND1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 116 comp-y: 101
and-gate HA1-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 114 comp-y: 200
or-gate HA1-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 154
not-gate HA1-NOT delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 351 comp-y: 158
and-gate HA2-AND1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 113 comp-y: 93
and-gate HA2-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 108 comp-y: 197
or-gate HA2-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 233 comp-y: 149
not-gate HA2-NOT delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 369 comp-y: 151
or-gate LAST-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 116 comp-y: 194
not-gate NOT-FIRSTX delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 251 comp-y: 295
not-gate NOT-FIRSTY delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 400
not-gate NOT-SECONDX delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 106 comp-y: 497
not-gate NOT-SECONDY delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 258 comp-y: 97
led SUM-OUTPUT manufacturer: " " color: RED comp-x: 249
  comp-y: 496
led CARRY-OUTPUT manufacturer: " " color: RED comp-x: 254
  comp-y: 190
subsystem ADDER3 is controls:
  HA1, HA2, LAST-OR, NOT-FIRSTX, NOT-FIRSTY, NOT-SECONDX,
  NOT-SECONDY, SUM-OUTPUT, CARRY-OUTPUT, SWITCH-1ADDEND,
  SWITCH-2ADDEND, SWITCH-3ADDEND
  imports:
    IN1 SIGNAL BOOLEAN HA1-AND1 ( OUT1 ADDER3 NOT-FIRSTX
      ) import-path: ( ADDER3 HA1 ) import-owner: HA1
      IN2 SIGNAL BOOLEAN HA1-AND1 ( OUT1 ADDER3 NOT-FIRSTY
        ) import-path: ( ADDER3 HA1 ) import-owner: HA1
      IN1 SIGNAL BOOLEAN HA1-AND2 ( OUT1 ADDER3 SWITCH-1ADDEND
        ) import-path: ( ADDER3 HA1 ) import-owner: HA1
      IN2 SIGNAL BOOLEAN HA1-AND2 ( OUT1 ADDER3 SWITCH-2ADDEND
        ) import-path: ( ADDER3 HA1 ) import-owner: HA1
      IN1 SIGNAL BOOLEAN HA1-OR ( OUT1 HA1 HA1-AND1
        ) import-path: ( ADDER3 HA1 ) import-owner: HA1
      IN2 SIGNAL BOOLEAN HA1-OR ( OUT1 HA1 HA1-AND2
        ) import-path: ( ADDER3 HA1 ) import-owner: HA1
      IN1 SIGNAL BOOLEAN HA1-NOT ( OUT1 HA1 HA1-OR
        ) import-path: ( ADDER3 HA1 ) import-owner: HA1
    IN1 SIGNAL BOOLEAN HA2-AND1 ( OUT1 ADDER3 NOT-SECONDX
      ) import-path: ( ADDER3 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-AND1 ( OUT1 ADDER3 NOT-SECONDY
      ) import-path: ( ADDER3 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-AND2 ( OUT1 HA1 HA1-NOT
      ) import-path: ( ADDER3 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-AND2 ( OUT1 ADDER3 SWITCH-3ADDEND
      ) import-path: ( ADDER3 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-OR ( OUT1 HA2 HA2-AND1
      ) import-path: ( ADDER3 HA2 ) import-owner: HA2
    IN2 SIGNAL BOOLEAN HA2-OR ( OUT1 HA2 HA2-AND2
      ) import-path: ( ADDER3 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN HA2-NOT ( OUT1 HA2 HA2-OR
      ) import-path: ( ADDER3 HA2 ) import-owner: HA2
    IN1 SIGNAL BOOLEAN CARRY-OUTPUT ( OUT1 ADDER3 LAST-OR
      ) import-path: ( ADDER3 ) import-owner: ADDER3
    IN1 SIGNAL BOOLEAN SUM-OUTPUT ( OUT1 HA2 HA2-NOT
      ) import-path: ( ADDER3 ) import-owner: ADDER3
    IN1 SIGNAL BOOLEAN NOT-SECONDY ( OUT1 ADDER3 SWITCH-3ADDEND
      ) import-path: ( ADDER3 ) import-owner: ADDER3
    IN1 SIGNAL BOOLEAN NOT-SECONDX ( OUT1 HA1 HA1-NOT
      ) import-path: ( ADDER3 ) import-owner: ADDER3
    IN1 SIGNAL BOOLEAN NOT-FIRSTY ( OUT1 ADDER3 SWITCH-2ADDEND
      ) import-path: ( ADDER3 ) import-owner: ADDER3
    IN1 SIGNAL BOOLEAN NOT-FIRSTX ( OUT1 ADDER3 SWITCH-1ADDEND
      ) import-path: ( ADDER3 ) import-owner: ADDER3
    IN2 SIGNAL BOOLEAN LAST-OR ( OUT1 HA1 HA1-AND2
      ) import-path: ( ADDER3 ) import-owner: ADDER3
    IN1 SIGNAL BOOLEAN LAST-OR ( OUT1 HA2 HA2-AND2
      ) import-path: ( ADDER3 ) import-owner: ADDER3
exports:
  OUT1 SIGNAL BOOLEAN HA1-AND1
    export-path: ( ADDER3 HA1 ) export-owner: HA1
    OUT1 SIGNAL BOOLEAN HA1-AND2
      export-path: ( ADDER3 HA1 ) export-owner: HA1
    OUT1 SIGNAL BOOLEAN HA1-OR
      export-path: ( ADDER3 HA1 ) export-owner: HA1
    OUT1 SIGNAL BOOLEAN HA1-NOT
      export-path: ( ADDER3 HA1 ) export-owner: HA1
    OUT1 SIGNAL BOOLEAN HA2-AND1
      export-path: ( ADDER3 HA2 ) export-owner: HA2
    OUT1 SIGNAL BOOLEAN HA2-AND2
      export-path: ( ADDER3 HA2 ) export-owner: HA2
    OUT1 SIGNAL BOOLEAN HA2-OR
      export-path: ( ADDER3 HA2 ) export-owner: HA2
    OUT1 SIGNAL BOOLEAN HA2-NOT
      export-path: ( ADDER3 HA2 ) export-owner: HA2
    OUT1 SIGNAL BOOLEAN SWITCH-3ADDEND
      export-path: ( ADDER3 ) export-owner: ADDER3
    OUT1 SIGNAL BOOLEAN SWITCH-2ADDEND
      export-path: ( ADDER3 ) export-owner: ADDER3
```

```
        OUT1 SIGNAL BOOLEAN SWITCH-1ADDEND
          export-path: ( ADDER3 ) export-owner: ADDER3
        OUT1 SIGNAL BOOLEAN NOT-SECONDY
          export-path: ( ADDER3 ) export-owner: ADDER3
        OUT1 SIGNAL BOOLEAN NOT-SECONDX
          export-path: ( ADDER3 ) export-owner: ADDER3
        OUT1 SIGNAL BOOLEAN NOT-FIRSTY
          export-path: ( ADDER3 ) export-owner: ADDER3
        OUT1 SIGNAL BOOLEAN NOT-FIRSTX
          export-path: ( ADDER3 ) export-owner: ADDER3
        OUT1 SIGNAL BOOLEAN LAST-OR
          export-path: ( ADDER3 ) export-owner: ADDER3
    initialize procedure: update procedure:
    update SWITCH-1ADDEND update SWITCH-2ADDEND
      update SWITCH-3ADDEND update NOT-FIRSTX update NOT-FIRSTY
      update HA1 update NOT-SECONDX update NOT-SECONDY update HA2
      update LAST-OR update CARRY-OUTPUT update SUM-OUTPUT
  and-gate AND-M0-A delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 571 comp-y: 755
  and-gate AND-M0-B delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 916 comp-y: 758
  and-gate AND-M1-A delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 569 comp-y: 636
  and-gate AND-M1-B delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 927 comp-y: 647
  and-gate AND-M2-A delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 567 comp-y: 553
  and-gate AND-M2-B delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 933 comp-y: 548
  and-gate AND-M3-A delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 567 comp-y: 468
  and-gate AND-M3-B delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 934 comp-y: 457
  and-gate AND-M4-A delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 767 comp-y: 360
  and-gate AND-M4-B delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 916 comp-y: 367
  and-gate AND-M5-A delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 753 comp-y: 256
  and-gate AND-M5-B delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 921 comp-y: 267
  and-gate AND-M6-A delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 765 comp-y: 157
  and-gate AND-M6-B delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 930 comp-y: 163
  and-gate AND-M7-A delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 767 comp-y: 57
  and-gate AND-M7-B delay: 0 not mil-spec manufacturer: " "
    power level: 0.0 comp-x: 931 comp-y: 47
  led M0 manufacturer: " " color: RED comp-x: 1037 comp-y: 754
  led M1 manufacturer: " " color: RED comp-x: 1035 comp-y: 650
```

```
led M2 manufacturer: " " color: RED comp-x: 1035 comp-y: 549
led M3 manufacturer: " " color: RED comp-x: 1035 comp-y: 463
led M4 manufacturer: " " color: RED comp-x: 1036 comp-y: 368
led M5 manufacturer: " " color: RED comp-x: 1031 comp-y: 264
led M6 manufacturer: " " color: RED comp-x: 1039 comp-y: 150
led M7 manufacturer: " " color: RED comp-x: 1050 comp-y: 44
not-gate NOT-X delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 730 comp-y: 596
not-gate NOT-Y delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 225 comp-y: 354
not-gate NOT-Z delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 219 comp-y: 166
subsystem DECODER-WITH-OUTPUT is controls:
  NOT-X, NOT-Y, NOT-Z, AND-M0-A, AND-M0-B, AND-M1-A,
  AND-M1-B, AND-M2-A, AND-M2-B, AND-M3-A, AND-M3-B, AND-M4-A,
  AND-M4-B, AND-M5-A, AND-M5-B, AND-M6-A, AND-M6-B, AND-M7-A,
  AND-M7-B, M0, M1, M2, M3, M4, M5, M6,
  M7 imports: exports: initialize procedure:
  update procedure:
  update NOT-X update NOT-Y update NOT-Z update AND-M0-A
    update AND-M0-B update AND-M1-A update AND-M1-B
    update AND-M2-A update AND-M2-B update AND-M3-A
    update AND-M3-B update AND-M4-A update AND-M4-B
    update AND-M5-A update AND-M5-B update AND-M6-A
    update AND-M6-B update AND-M7-A update AND-M7-B update M0
    update M1 update M2 update M3 update M4 update M5 update M6
    update M7
subsystem HA1 is controls:
  HA1-AND1, HA1-AND2, HA1-OR, HA1-NOT imports: exports:
  initialize procedure: update procedure:
  update HA1-AND1 update HA1-AND2 update HA1-OR update HA1-NOT
  comp-x: 460 comp-y: 297
subsystem HA2 is controls:
  HA2-AND1, HA2-AND2, HA2-OR, HA2-NOT imports: exports:
  initialize procedure: update procedure:
  update HA2-AND1 update HA2-AND2 update HA2-OR update HA2-NOT
  comp-x: 468 comp-y: 88
```

C.6 BCD-ADDER Application

This application tests a subsystem implementation of a BCD-adder using primitive gates and half-adders.

```
application definition BCD-ADDER
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
application BCD-ADDER is controls: DRIVE-BCD-ADDER
  update procedure: update DRIVE-BCD-ADDER
subsystem DRIVE-BCD-ADDER is controls:
  BCD-ADD, ADDEND1, ADDEND2, ADDEND3, ADDEND4, CARRY-IN,
NULL, AUGEND1, AUGEND2, AUGEND3, AUGEND4, CARRY-OUT, S8,
S4, S2, S1
imports:
  IN2 SIGNAL BOOLEAN BCD-ORA ( OUT1 FOUR-BIT-1 CARRY-4-1
    ) import-path: ( DRIVE-BCD-ADDER BCD-ADD ) import-owner:
        BCD-ADD
    IN1 SIGNAL BOOLEAN BCD-ORA ( OUT1 BCD-ADD BCD-AND1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD ) import-owner:
          BCD-ADD
    IN2 SIGNAL BOOLEAN BCD-OR ( OUT1 BCD-ADD BCD-AND2
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD ) import-owner:
          BCD-ADD
    IN1 SIGNAL BOOLEAN BCD-OR ( OUT1 BCD-ADD BCD-ORA
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD ) import-owner:
          BCD-ADD
    IN2 SIGNAL BOOLEAN BCD-AND2 ( S FOUR-BIT-1 HA2B-1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD ) import-owner:
          BCD-ADD
    IN1 SIGNAL BOOLEAN BCD-AND2 ( S FOUR-BIT-1 HA4B-1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD ) import-owner:
          BCD-ADD
    IN2 SIGNAL BOOLEAN BCD-AND1 ( S FOUR-BIT-1 HA3B-1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD ) import-owner:
          BCD-ADD
    IN1 SIGNAL BOOLEAN BCD-AND1 ( S FOUR-BIT-1 HA4B-1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD ) import-owner:
          BCD-ADD
    IN1 SIGNAL BOOLEAN HA1A-1 ( OUT1 DRIVE-BCD-ADDER CARRY-IN
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
          ) import-owner: FOUR-BIT-1
    IN2 SIGNAL BOOLEAN HA1A-1 ( OUT1 DRIVE-BCD-ADDER ADDEND1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
          ) import-owner: FOUR-BIT-1
    IN1 SIGNAL BOOLEAN HA1B-1 ( S FOUR-BIT-1 HA1A-1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
          ) import-owner: FOUR-BIT-1
    IN2 SIGNAL BOOLEAN HA1B-1 ( OUT1 DRIVE-BCD-ADDER AUGEND1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
          ) import-owner: FOUR-BIT-1
    IN1 SIGNAL BOOLEAN HA2A-1 ( OUT1 FOUR-BIT-1 CARRY-1-1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
          ) import-owner: FOUR-BIT-1
    IN2 SIGNAL BOOLEAN HA2A-1 ( OUT1 DRIVE-BCD-ADDER ADDEND2
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
          ) import-owner: FOUR-BIT-1
    IN1 SIGNAL BOOLEAN HA2B-1 ( S FOUR-BIT-1 HA2A-1
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
          ) import-owner: FOUR-BIT-1
    IN2 SIGNAL BOOLEAN HA2B-1 ( OUT1 DRIVE-BCD-ADDER AUGEND2
      ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
          ) import-owner: FOUR-BIT-1
```

```
IN1 SIGNAL BOOLEAN HA3A-1 ( OUT1 FOUR-BIT-1 CARRY-2-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN2 SIGNAL BOOLEAN HA3A-1 ( OUT1 DRIVE-BCD-ADDER ADDEND3
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN1 SIGNAL BOOLEAN HA3B-1 ( S FOUR-BIT-1 HA3A-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN2 SIGNAL BOOLEAN HA3B-1 ( OUT1 DRIVE-BCD-ADDER AUGEND3
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN1 SIGNAL BOOLEAN HA4A-1 ( OUT1 FOUR-BIT-1 CARRY-3-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN2 SIGNAL BOOLEAN HA4A-1 ( OUT1 DRIVE-BCD-ADDER ADDEND4
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN1 SIGNAL BOOLEAN HA4B-1 ( S FOUR-BIT-1 HA4A-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN2 SIGNAL BOOLEAN HA4B-1 ( OUT1 DRIVE-BCD-ADDER AUGEND4
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN1 SIGNAL BOOLEAN CARRY-1-1 ( C FOUR-BIT-1 HA1A-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN2 SIGNAL BOOLEAN CARRY-1-1 ( C FOUR-BIT-1 HA1B-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN1 SIGNAL BOOLEAN CARRY-2-1 ( C FOUR-BIT-1 HA2A-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN2 SIGNAL BOOLEAN CARRY-2-1 ( C FOUR-BIT-1 HA2B-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN1 SIGNAL BOOLEAN CARRY-3-1 ( C FOUR-BIT-1 HA3B-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN2 SIGNAL BOOLEAN CARRY-3-1 ( C FOUR-BIT-1 HA3A-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN1 SIGNAL BOOLEAN CARRY-4-1 ( C FOUR-BIT-1 HA4B-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN2 SIGNAL BOOLEAN CARRY-4-1 ( C FOUR-BIT-1 HA4A-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
      ) import-owner: FOUR-BIT-1
IN1 SIGNAL BOOLEAN HA1A-2 ( OUT1 DRIVE-BCD-ADDER NULL
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
```

```
IN2 SIGNAL BOOLEAN HA1A-2 ( S FOUR-BIT-1 HA1B-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN1 SIGNAL BOOLEAN HA1B-2 ( S FOUR-BIT-2 HA1A-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN2 SIGNAL BOOLEAN HA1B-2 ( OUT1 DRIVE-BCD-ADDER NULL
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN1 SIGNAL BOOLEAN HA2A-2 ( OUT1 FOUR-BIT-2 CARRY-1-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN2 SIGNAL BOOLEAN HA2A-2 ( S FOUR-BIT-1 HA2B-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN1 SIGNAL BOOLEAN HA2B-2 ( S FOUR-BIT-2 HA2A-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN2 SIGNAL BOOLEAN HA2B-2 ( OUT1 BCD-ADD BCD-OR
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN1 SIGNAL BOOLEAN HA3A-2 ( OUT1 FOUR-BIT-2 CARRY-2-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN2 SIGNAL BOOLEAN HA3A-2 ( S FOUR-BIT-1 HA3B-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN1 SIGNAL BOOLEAN HA3B-2 ( S FOUR-BIT-2 HA3A-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN2 SIGNAL BOOLEAN HA3B-2 ( OUT1 BCD-ADD BCD-OR
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN1 SIGNAL BOOLEAN HA4A-2 ( OUT1 FOUR-BIT-2 CARRY-3-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN2 SIGNAL BOOLEAN HA4A-2 ( S FOUR-BIT-1 HA4B-1
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN1 SIGNAL BOOLEAN HA4B-2 ( S FOUR-BIT-2 HA4A-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN2 SIGNAL BOOLEAN HA4B-2 ( OUT1 DRIVE-BCD-ADDER NULL
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN1 SIGNAL BOOLEAN CARRY-1-2 ( C FOUR-BIT-2 HA1A-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
IN2 SIGNAL BOOLEAN CARRY-1-2 ( C FOUR-BIT-2 HA1B-2
  ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
      ) import-owner: FOUR-BIT-2
      IN1 SIGNAL BOOLEAN CARRY-2-2 ( C FOUR-BIT-2 HA2A-2
        ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
            ) import-owner: FOUR-BIT-2
      IN2 SIGNAL BOOLEAN CARRY-2-2 ( C FOUR-BIT-2 HA2B-2
        ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
            ) import-owner: FOUR-BIT-2
      IN1 SIGNAL BOOLEAN CARRY-3-2 ( C FOUR-BIT-2 HA3B-2
        ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
            ) import-owner: FOUR-BIT-2
      IN2 SIGNAL BOOLEAN CARRY-3-2 ( C FOUR-BIT-2 HA3A-2
        ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
            ) import-owner: FOUR-BIT-2
      IN1 SIGNAL BOOLEAN CARRY-4-2 ( C FOUR-BIT-2 HA4B-2
        ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
            ) import-owner: FOUR-BIT-2
      IN2 SIGNAL BOOLEAN CARRY-4-2 ( C FOUR-BIT-2 HA4A-2
        ) import-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
            ) import-owner: FOUR-BIT-2
      IN1 SIGNAL BOOLEAN S1 ( S FOUR-BIT-2 HA1B-2
        ) import-path: ( DRIVE-BCD-ADDER ) import-owner:
            DRIVE-BCD-ADDER
      IN1 SIGNAL BOOLEAN S2 ( S FOUR-BIT-2 HA2B-2
        ) import-path: ( DRIVE-BCD-ADDER ) import-owner:
            DRIVE-BCD-ADDER
      IN1 SIGNAL BOOLEAN S4 ( S FOUR-BIT-2 HA3B-2
        ) import-path: ( DRIVE-BCD-ADDER ) import-owner:
            DRIVE-BCD-ADDER
      IN1 SIGNAL BOOLEAN S8 ( S FOUR-BIT-2 HA4B-2
        ) import-path: ( DRIVE-BCD-ADDER ) import-owner:
            DRIVE-BCD-ADDER
      IN1 SIGNAL BOOLEAN CARRY-OUT ( OUT1 BCD-ADD BCD-OR
        ) import-path: ( DRIVE-BCD-ADDER ) import-owner:
            DRIVE-BCD-ADDER
exports:
  OUT1 SIGNAL BOOLEAN BCD-ORA
    export-path: ( DRIVE-BCD-ADDER BCD-ADD ) export-owner:
      BCD-ADD
    OUT1 SIGNAL BOOLEAN BCD-OR
      export-path: ( DRIVE-BCD-ADDER BCD-ADD ) export-owner:
        BCD-ADD
    OUT1 SIGNAL BOOLEAN BCD-AND2
      export-path: ( DRIVE-BCD-ADDER BCD-ADD ) export-owner:
        BCD-ADD
    OUT1 SIGNAL BOOLEAN BCD-AND1
      export-path: ( DRIVE-BCD-ADDER BCD-ADD ) export-owner:
        BCD-ADD
    S SIGNAL BOOLEAN HA1A-1
      export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
        ) export-owner: FOUR-BIT-1
    C SIGNAL BOOLEAN HA1A-1
      export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
S SIGNAL BOOLEAN HA1B-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
C SIGNAL BOOLEAN HA1B-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
S SIGNAL BOOLEAN HA2A-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
C SIGNAL BOOLEAN HA2A-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
S SIGNAL BOOLEAN HA2B-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
C SIGNAL BOOLEAN HA2B-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
S SIGNAL BOOLEAN HA3A-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
C SIGNAL BOOLEAN HA3A-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
S SIGNAL BOOLEAN HA3B-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
C SIGNAL BOOLEAN HA3B-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
S SIGNAL BOOLEAN HA4A-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
C SIGNAL BOOLEAN HA4A-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
S SIGNAL BOOLEAN HA4B-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
C SIGNAL BOOLEAN HA4B-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
OUT1 SIGNAL BOOLEAN CARRY-1-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
OUT1 SIGNAL BOOLEAN CARRY-2-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
OUT1 SIGNAL BOOLEAN CARRY-3-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
OUT1 SIGNAL BOOLEAN CARRY-4-1
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-1
    ) export-owner: FOUR-BIT-1
S SIGNAL BOOLEAN HA1A-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
C SIGNAL BOOLEAN HA1A-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
S SIGNAL BOOLEAN HA1B-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
C SIGNAL BOOLEAN HA1B-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
S SIGNAL BOOLEAN HA2A-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
C SIGNAL BOOLEAN HA2A-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
S SIGNAL BOOLEAN HA2B-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
C SIGNAL BOOLEAN HA2B-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
S SIGNAL BOOLEAN HA3A-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
C SIGNAL BOOLEAN HA3A-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
S SIGNAL BOOLEAN HA3B-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
C SIGNAL BOOLEAN HA3B-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
S SIGNAL BOOLEAN HA4A-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
C SIGNAL BOOLEAN HA4A-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
S SIGNAL BOOLEAN HA4B-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
    ) export-owner: FOUR-BIT-2
C SIGNAL BOOLEAN HA4B-2
  export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
          ) export-owner: FOUR-BIT-2
      OUT1 SIGNAL BOOLEAN CARRY-1-2
        export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
          ) export-owner: FOUR-BIT-2
      OUT1 SIGNAL BOOLEAN CARRY-2-2
        export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
          ) export-owner: FOUR-BIT-2
      OUT1 SIGNAL BOOLEAN CARRY-3-2
        export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
          ) export-owner: FOUR-BIT-2
      OUT1 SIGNAL BOOLEAN CARRY-4-2
        export-path: ( DRIVE-BCD-ADDER BCD-ADD FOUR-BIT-2
          ) export-owner: FOUR-BIT-2
      OUT1 SIGNAL BOOLEAN NULL
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN AUGEND4
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN AUGEND3
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN AUGEND2
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN AUGEND1
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN CARRY-IN
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN ADDEND4
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN ADDEND3
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN ADDEND2
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
      OUT1 SIGNAL BOOLEAN ADDEND1
        export-path: ( DRIVE-BCD-ADDER ) export-owner:
          DRIVE-BCD-ADDER
  initialize procedure: update procedure:
  update NULL update CARRY-IN update ADDEND1 update ADDEND2
    update ADDEND3 update ADDEND4 update AUGEND1 update AUGEND2
    update AUGEND3 update AUGEND4 update BCD-ADD
    update CARRY-OUT update S8 update S4 update S2 update S1
subsystem BCD-ADD is controls:
  FOUR-BIT-1, FOUR-BIT-2, BCD-AND1, BCD-AND2, BCD-OR, BCD-ORA
  imports: exports: initialize procedure: update procedure:
```

```
  update FOUR-BIT-1 update BCD-AND1 update BCD-AND2
    update BCD-ORA update BCD-OR update FOUR-BIT-2
  comp-x: 183 comp-y: 495
subsystem FOUR-BIT-1 is controls:
  HA1A-1, HA1B-1, HA2A-1, HA2B-1, HA3A-1, HA3B-1, HA4A-1,
  HA4B-1, CARRY-1-1, CARRY-2-1, CARRY-3-1, CARRY-4-1
  imports: exports: initialize procedure: update procedure:
  update HA1A-1 update HA1B-1 update CARRY-1-1 update HA2A-1
    update HA2B-1 update CARRY-2-1 update HA3A-1 update HA3B-1
    update CARRY-3-1 update HA4A-1 update HA4B-1
    update CARRY-4-1
  comp-x: 469 comp-y: 347
subsystem FOUR-BIT-2 is controls:
  HA1A-2, HA1B-2, HA2A-2, HA2B-2, HA3A-2, HA3B-2, HA4A-2,
  HA4B-2, CARRY-1-2, CARRY-2-2, CARRY-3-2, CARRY-4-2
  imports: exports: initialize procedure: update procedure:
  update HA1A-2 update HA1B-2 update CARRY-1-2 update HA2A-2
    update HA2B-2 update CARRY-2-2 update HA3A-2 update HA3B-2
    update CARRY-3-2 update HA4A-2 update HA4B-2
    update CARRY-4-2
  comp-x: 167 comp-y: 348
half-adder HA1A-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 95 comp-y: 549
half-adder HA1B-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 260 comp-y: 490
or-gate CARRY-1-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 397 comp-y: 551
half-adder HA2A-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 504 comp-y: 426
half-adder HA2B-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 604 comp-y: 549
or-gate CARRY-2-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 750 comp-y: 503
half-adder HA3A-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 789 comp-y: 342
half-adder HA3B-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 300
or-gate CARRY-3-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 120 comp-y: 200
half-adder HA4A-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 200
half-adder HA4B-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 100
or-gate CARRY-4-1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 120 comp-y: 100
half-adder HA1A-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 360 comp-y: 400
half-adder HA1B-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 360 comp-y: 300
or-gate CARRY-1-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 120 comp-y: 400
```

```
half-adder HA2A-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 360 comp-y: 200
half-adder HA2B-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 360 comp-y: 100
or-gate CARRY-2-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 120 comp-y: 300
half-adder HA3A-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 400
half-adder HA3B-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 300
or-gate CARRY-3-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 120 comp-y: 200
half-adder HA4A-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 200
half-adder HA4B-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 240 comp-y: 100
or-gate CARRY-4-2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 120 comp-y: 100
and-gate BCD-AND1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 103 comp-y: 245
and-gate BCD-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 116 comp-y: 149
or-gate BCD-ORA delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 233 comp-y: 250
or-gate BCD-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 376 comp-y: 212
switch NULL delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 360 comp-y: 200
switch CARRY-IN delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 360 comp-y: 300
switch ADDEND1 delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 480 comp-y: 300
switch ADDEND2 delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 480 comp-y: 200
switch ADDEND3 delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 480 comp-y: 100
switch ADDEND4 delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 360 comp-y: 400
switch AUGEND1 delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 360 comp-y: 100
switch AUGEND2 delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 240 comp-y: 400
switch AUGEND3 delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 240 comp-y: 300
switch AUGEND4 delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 240 comp-y: 200
led CARRY-OUT manufacturer: " " color: RED comp-x: 240
  comp-y: 100
led S8 manufacturer: " " color: RED comp-x: 120 comp-y: 400
led S4 manufacturer: " " color: RED comp-x: 120 comp-y: 300
led S2 manufacturer: " " color: RED comp-x: 120 comp-y: 200
led S1 manufacturer: " " color: RED comp-x: 120 comp-y: 100
```

C.7 BINARY-ARRAY-MULTIPLIER1 Application

This application implements a binary array multiplier as a top level subsystem, composed of "and" gates, "not" gates, switches, LEDs, and 2 half-adder subsystems, each of which is composed of "and" gates, "or" gates, and "not" gates.

```
application definition BINARY-ARRAY-MULTIPLIER1
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
application BINARY-ARRAY-MULTIPLIER1 is controls:
  BIN-AR-MULT update procedure: update BIN-AR-MULT
switch A-ZERO delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 105 comp-y: 517
switch A-ONE delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 106 comp-y: 412
switch B-ZERO delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 104 comp-y: 308
switch B-ONE delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 103 comp-y: 194
led C-ZERO manufacturer: " " color: RED comp-x: 572
  comp-y: 301
led C-ONE manufacturer: " " color: RED comp-x: 583
  comp-y: 399
led C-TWO manufacturer: " " color: RED comp-x: 591
  comp-y: 528
led C-THREE manufacturer: " " color: RED comp-x: 581
  comp-y: 183
and-gate AND-A1-B1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 282 comp-y: 178
and-gate AND-A1-B0 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 285 comp-y: 404
and-gate AND-A0-B1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 296 comp-y: 511
and-gate AND-A0-B0 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 280 comp-y: 297
and-gate HA-1-AND1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 107 comp-y: 68
and-gate HA-1-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 106 comp-y: 153
or-gate HA-1-OR delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 244 comp-y: 122
not-gate HA-1-NOT delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 363 comp-y: 126
and-gate HA-2-AND1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 122 comp-y: 99
and-gate HA-2-AND2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 113 comp-y: 197
or-gate HA-2-OR delay: 0 not mil-spec manufacturer: " "
```

```
  power level: 0.0 comp-x: 237 comp-y: 156
not-gate HA-2-NOT delay: 0 is mil-spec manufacturer: " "
  power level: 0.0 comp-x: 352 comp-y: 160
not-gate NOT-X-HA1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 440 comp-y: 411
not-gate NOT-Y-HA1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 439 comp-y: 510
not-gate NOT-X-HA2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 116 comp-y: 72
not-gate NOT-Y-HA2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 439 comp-y: 176
subsystem BIN-AR-MULT is controls:
  HA-1, HA-2, AND-A1-B1, AND-A1-B0, AND-A0-B1, AND-A0-B0,
  A-ZERO, A-ONE, B-ZERO, B-ONE, C-ZERO, C-ONE, C-TWO,
  C-THREE, NOT-X-HA1, NOT-Y-HA1, NOT-X-HA2, NOT-Y-HA2
  imports:
    IN1 SIGNAL BOOLEAN HA-1-AND1 ( OUT1 BIN-AR-MULT NOT-X-HA1
      ) import-path: ( BIN-AR-MULT HA-1 ) import-owner: HA-1
      IN2 SIGNAL BOOLEAN HA-1-AND1 ( OUT1 BIN-AR-MULT NOT-Y-HA1
        ) import-path: ( BIN-AR-MULT HA-1 ) import-owner: HA-1
      IN1 SIGNAL BOOLEAN HA-1-AND2 ( OUT1 BIN-AR-MULT AND-A1-B0
        ) import-path: ( BIN-AR-MULT HA-1 ) import-owner: HA-1
      IN2 SIGNAL BOOLEAN HA-1-AND2 ( OUT1 BIN-AR-MULT AND-A0-B1
        ) import-path: ( BIN-AR-MULT HA-1 ) import-owner: HA-1
      IN1 SIGNAL BOOLEAN HA-1-OR ( OUT1 HA-1 HA-1-AND1
        ) import-path: ( BIN-AR-MULT HA-1 ) import-owner: HA-1
      IN2 SIGNAL BOOLEAN HA-1-OR ( OUT1 HA-1 HA-1-AND2
        ) import-path: ( BIN-AR-MULT HA-1 ) import-owner: HA-1
      IN1 SIGNAL BOOLEAN HA-1-NOT ( OUT1 HA-1 HA-1-OR
        ) import-path: ( BIN-AR-MULT HA-1 ) import-owner: HA-1
      IN1 SIGNAL BOOLEAN HA-2-AND1 ( OUT1 BIN-AR-MULT NOT-X-HA2
        ) import-path: ( BIN-AR-MULT HA-2 ) import-owner: HA-2
      IN2 SIGNAL BOOLEAN HA-2-AND1 ( OUT1 BIN-AR-MULT NOT-Y-HA2
        ) import-path: ( BIN-AR-MULT HA-2 ) import-owner: HA-2
      IN1 SIGNAL BOOLEAN HA-2-AND2 ( OUT1 HA-1 HA-1-AND2
        ) import-path: ( BIN-AR-MULT HA-2 ) import-owner: HA-2
      IN2 SIGNAL BOOLEAN HA-2-AND2 ( OUT1 BIN-AR-MULT AND-A1-B1
        ) import-path: ( BIN-AR-MULT HA-2 ) import-owner: HA-2
      IN1 SIGNAL BOOLEAN HA-2-OR ( OUT1 HA-2 HA-2-AND1
        ) import-path: ( BIN-AR-MULT HA-2 ) import-owner: HA-2
      IN2 SIGNAL BOOLEAN HA-2-OR ( OUT1 HA-2 HA-2-AND2
        ) import-path: ( BIN-AR-MULT HA-2 ) import-owner: HA-2
      IN1 SIGNAL BOOLEAN HA-2-NOT ( OUT1 HA-2 HA-2-OR
        ) import-path: ( BIN-AR-MULT HA-2 ) import-owner: HA-2
      IN1 SIGNAL BOOLEAN NOT-Y-HA2 ( OUT1 BIN-AR-MULT AND-A1-B1
        ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
      IN1 SIGNAL BOOLEAN NOT-X-HA2 ( OUT1 HA-1 HA-1-AND2
        ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
      IN1 SIGNAL BOOLEAN NOT-Y-HA1 ( OUT1 BIN-AR-MULT AND-A0-B1
        ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
      IN1 SIGNAL BOOLEAN NOT-X-HA1 ( OUT1 BIN-AR-MULT AND-A1-B0
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN1 SIGNAL BOOLEAN C-THREE ( OUT1 HA-2 HA-2-AND2
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN1 SIGNAL BOOLEAN C-TWO ( OUT1 HA-2 HA-2-NOT
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN1 SIGNAL BOOLEAN C-ONE ( OUT1 HA-1 HA-1-NOT
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN1 SIGNAL BOOLEAN C-ZERO ( OUT1 BIN-AR-MULT AND-A0-B0
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN2 SIGNAL BOOLEAN AND-A0-B0 ( OUT1 BIN-AR-MULT B-ZERO
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN1 SIGNAL BOOLEAN AND-A0-B0 ( OUT1 BIN-AR-MULT A-ZERO
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN2 SIGNAL BOOLEAN AND-A0-B1 ( OUT1 BIN-AR-MULT B-ONE
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN1 SIGNAL BOOLEAN AND-A0-B1 ( OUT1 BIN-AR-MULT A-ZERO
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN2 SIGNAL BOOLEAN AND-A1-B0 ( OUT1 BIN-AR-MULT B-ZERO
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN1 SIGNAL BOOLEAN AND-A1-B0 ( OUT1 BIN-AR-MULT A-ONE
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN2 SIGNAL BOOLEAN AND-A1-B1 ( OUT1 BIN-AR-MULT B-ONE
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
    IN1 SIGNAL BOOLEAN AND-A1-B1 ( OUT1 BIN-AR-MULT A-ONE
      ) import-path: ( BIN-AR-MULT ) import-owner: BIN-AR-MULT
exports:
  OUT1 SIGNAL BOOLEAN HA-1-AND1
    export-path: ( BIN-AR-MULT HA-1 ) export-owner: HA-1
    OUT1 SIGNAL BOOLEAN HA-1-AND2
      export-path: ( BIN-AR-MULT HA-1 ) export-owner: HA-1
    OUT1 SIGNAL BOOLEAN HA-1-OR
      export-path: ( BIN-AR-MULT HA-1 ) export-owner: HA-1
    OUT1 SIGNAL BOOLEAN HA-1-NOT
      export-path: ( BIN-AR-MULT HA-1 ) export-owner: HA-1
    OUT1 SIGNAL BOOLEAN HA-2-AND1
      export-path: ( BIN-AR-MULT HA-2 ) export-owner: HA-2
    OUT1 SIGNAL BOOLEAN HA-2-AND2
      export-path: ( BIN-AR-MULT HA-2 ) export-owner: HA-2
    OUT1 SIGNAL BOOLEAN HA-2-OR
      export-path: ( BIN-AR-MULT HA-2 ) export-owner: HA-2
    OUT1 SIGNAL BOOLEAN HA-2-NOT
      export-path: ( BIN-AR-MULT HA-2 ) export-owner: HA-2
    OUT1 SIGNAL BOOLEAN NOT-Y-HA2
      export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
    OUT1 SIGNAL BOOLEAN NOT-X-HA2
      export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
    OUT1 SIGNAL BOOLEAN NOT-Y-HA1
      export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
    OUT1 SIGNAL BOOLEAN NOT-X-HA1
      export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
    OUT1 SIGNAL BOOLEAN B-ONE
```

```
        export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
      OUT1 SIGNAL BOOLEAN B-ZERO
        export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
      OUT1 SIGNAL BOOLEAN A-ONE
        export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
      OUT1 SIGNAL BOOLEAN A-ZERO
        export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
      OUT1 SIGNAL BOOLEAN AND-A0-B0
        export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
      OUT1 SIGNAL BOOLEAN AND-A0-B1
        export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
      OUT1 SIGNAL BOOLEAN AND-A1-B0
        export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
      OUT1 SIGNAL BOOLEAN AND-A1-B1
        export-path: ( BIN-AR-MULT ) export-owner: BIN-AR-MULT
  initialize procedure: update procedure:
  update A-ZERO update A-ONE update B-ZERO update B-ONE
    update AND-A0-B1 update AND-A0-B0 update AND-A1-B1
    update AND-A1-B0 update NOT-X-HA1 update NOT-Y-HA1
    update NOT-X-HA2 update NOT-Y-HA2 update HA-1 update HA-2
    update C-ZERO update C-ONE update C-TWO update C-THREE
subsystem HA-1 is controls:
  HA-1-AND1, HA-1-AND2, HA-1-OR, HA-1-NOT imports: exports:
  initialize procedure: update procedure:
  update HA-1-AND1 update HA-1-AND2 update HA-1-OR
    update HA-1-NOT
  comp-x: 442 comp-y: 615
subsystem HA-2 is controls:
  HA-2-AND1, HA-2-AND2, HA-2-OR, HA-2-NOT imports: exports:
  initialize procedure: update procedure:
  update HA-2-AND1 update HA-2-AND2 update HA-2-OR
    update HA-2-NOT
  comp-x: 157 comp-y: 613
```

C.8 BINARY-ARRAY-MULTIPLIER2 Application

This application implements a binary array multiplier similar to the previous application, except that primitive half-adders are used instead of lower-level subsystems composed of primitive gates.

```
application definition BINARY-ARRAY-MULTIPLIER2
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
application BINARY-ARRAY-MULTIPLIER2 is controls: DO-BAM
  update procedure: update DO-BAM
and-gate A0B0 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 119 comp-y: 395
and-gate A1B0 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 118 comp-y: 100
and-gate A0B1 delay: 0 not mil-spec manufacturer: " "
```

```
  power level: 0.0 comp-x: 119 comp-y: 198
and-gate A1B1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 120 comp-y: 300
half-adder HA-PRIM1 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 367 comp-y: 250
half-adder HA-PRIM2 delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 241 comp-y: 154
subsystem BINARY-ARRAY-MULT is controls:
  A0B0, A1B0, A0B1, A1B1, HA-PRIM1, HA-PRIM2 imports: exports:
  initialize procedure: update procedure:
  update A0B1 update A0B0 update A1B1 update A1B0
    update HA-PRIM2 update HA-PRIM1
switch A-ZERO delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 360 comp-y: 200
switch A-ONE delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 360 comp-y: 100
switch B-ZERO delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 240 comp-y: 300
switch B-ONE delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 240 comp-y: 200
led C-ZERO manufacturer: " " color: RED comp-x: 240
  comp-y: 100
led C-ONE manufacturer: " " color: RED comp-x: 120
  comp-y: 300
led C-TWO manufacturer: " " color: RED comp-x: 120
  comp-y: 200
led C-THREE manufacturer: " " color: RED comp-x: 120
  comp-y: 100
subsystem DO-BAM is controls:
  A-ZERO, A-ONE, B-ZERO, B-ONE, C-ZERO, C-ONE, C-TWO,
  C-THREE, BINARY-ARRAY-MULT
  imports:
    IN2 SIGNAL BOOLEAN HA-PRIM2 ( OUT1 BINARY-ARRAY-MULT A1B0
      ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
          BINARY-ARRAY-MULT
      IN1 SIGNAL BOOLEAN HA-PRIM2 ( OUT1 BINARY-ARRAY-MULT A0B1
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
            BINARY-ARRAY-MULT
      IN2 SIGNAL BOOLEAN HA-PRIM1 ( C BINARY-ARRAY-MULT HA-PRIM2
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
            BINARY-ARRAY-MULT
      IN1 SIGNAL BOOLEAN HA-PRIM1 ( OUT1 BINARY-ARRAY-MULT A1B1
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
            BINARY-ARRAY-MULT
      IN2 SIGNAL BOOLEAN A1B1 ( OUT1 DO-BAM B-ONE
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
            BINARY-ARRAY-MULT
      IN1 SIGNAL BOOLEAN A1B1 ( OUT1 DO-BAM A-ONE
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
            BINARY-ARRAY-MULT
      IN2 SIGNAL BOOLEAN A0B1 ( OUT1 DO-BAM B-ONE
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
          BINARY-ARRAY-MULT
      IN1 SIGNAL BOOLEAN A0B1 ( OUT1 DO-BAM A-ZERO
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
          BINARY-ARRAY-MULT
      IN2 SIGNAL BOOLEAN A1B0 ( OUT1 DO-BAM B-ZERO
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
          BINARY-ARRAY-MULT
      IN1 SIGNAL BOOLEAN A1B0 ( OUT1 DO-BAM A-ONE
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
          BINARY-ARRAY-MULT
      IN2 SIGNAL BOOLEAN A0B0 ( OUT1 DO-BAM B-ZERO
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
          BINARY-ARRAY-MULT
      IN1 SIGNAL BOOLEAN A0B0 ( OUT1 DO-BAM A-ZERO
        ) import-path: ( DO-BAM BINARY-ARRAY-MULT ) import-owner:
          BINARY-ARRAY-MULT
      IN1 SIGNAL BOOLEAN C-THREE ( C BINARY-ARRAY-MULT HA-PRIM1
        ) import-path: ( DO-BAM ) import-owner: DO-BAM
      IN1 SIGNAL BOOLEAN C-TWO ( S BINARY-ARRAY-MULT HA-PRIM1
        ) import-path: ( DO-BAM ) import-owner: DO-BAM
      IN1 SIGNAL BOOLEAN C-ONE ( S BINARY-ARRAY-MULT HA-PRIM2
        ) import-path: ( DO-BAM ) import-owner: DO-BAM
      IN1 SIGNAL BOOLEAN C-ZERO ( OUT1 BINARY-ARRAY-MULT A0B1
        ) import-path: ( DO-BAM ) import-owner: DO-BAM
exports:
  C SIGNAL BOOLEAN HA-PRIM2
    export-path: ( DO-BAM BINARY-ARRAY-MULT ) export-owner:
      BINARY-ARRAY-MULT
    S SIGNAL BOOLEAN HA-PRIM2
      export-path: ( DO-BAM BINARY-ARRAY-MULT ) export-owner:
        BINARY-ARRAY-MULT
    C SIGNAL BOOLEAN HA-PRIM1
      export-path: ( DO-BAM BINARY-ARRAY-MULT ) export-owner:
        BINARY-ARRAY-MULT
    S SIGNAL BOOLEAN HA-PRIM1
      export-path: ( DO-BAM BINARY-ARRAY-MULT ) export-owner:
        BINARY-ARRAY-MULT
    OUT1 SIGNAL BOOLEAN A1B1
      export-path: ( DO-BAM BINARY-ARRAY-MULT ) export-owner:
        BINARY-ARRAY-MULT
    OUT1 SIGNAL BOOLEAN A0B1
      export-path: ( DO-BAM BINARY-ARRAY-MULT ) export-owner:
        BINARY-ARRAY-MULT
    OUT1 SIGNAL BOOLEAN A1B0
      export-path: ( DO-BAM BINARY-ARRAY-MULT ) export-owner:
        BINARY-ARRAY-MULT
    OUT1 SIGNAL BOOLEAN A0B0
      export-path: ( DO-BAM BINARY-ARRAY-MULT ) export-owner:
        BINARY-ARRAY-MULT
    OUT1 SIGNAL BOOLEAN B-ONE
```

```
      export-path: ( DO-BAM ) export-owner: DO-BAM
    OUT1 SIGNAL BOOLEAN B-ZERO
      export-path: ( DO-BAM ) export-owner: DO-BAM
    OUT1 SIGNAL BOOLEAN A-ONE
      export-path: ( DO-BAM ) export-owner: DO-BAM
    OUT1 SIGNAL BOOLEAN A-ZERO
      export-path: ( DO-BAM ) export-owner: DO-BAM
  initialize procedure: update procedure:
  update A-ONE update A-ZERO update B-ONE update B-ZERO
    update BINARY-ARRAY-MULT update C-THREE update C-TWO
    update C-ONE update C-ZERO
```

C.9 THREE-TO-EIGHT-DECODER Application

This application composes "and" gates and "not" gates into a subsystem implementation of a 3-to-8 decoder.

```
application definition THREE-TO-EIGHT-DECODER
execution-mode: NON-EVENT-DRIVEN-SEQUENTIAL
application THREE-TO-EIGHT-DECODER is controls: DRIVER
  update procedure: update DRIVER
switch SWITCH-IN-X delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 100 comp-y: 100
switch SWITCH-IN-Y delay: 0 not debounced manufacturer: " "
  position: ON comp-x: 100 comp-y: 200
switch SWITCH-IN-Z delay: 0 not debounced manufacturer: " "
  position: OFF comp-x: 100 comp-y: 300
led M0 manufacturer: " " color: RED comp-x: 300 comp-y: 100
led M1 manufacturer: " " color: RED comp-x: 300 comp-y: 200
led M2 manufacturer: " " color: RED comp-x: 300 comp-y: 300
led M3 manufacturer: " " color: RED comp-x: 300 comp-y: 400
led M4 manufacturer: " " color: RED comp-x: 300 comp-y: 500
led M5 manufacturer: " " color: RED comp-x: 300 comp-y: 600
led M6 manufacturer: " " color: RED comp-x: 300 comp-y: 700
led M7 manufacturer: " " color: RED comp-x: 300 comp-y: 800
subsystem DRIVER is controls:
  SWITCH-IN-X, SWITCH-IN-Y, SWITCH-IN-Z, M0, M1, M2, M3, M4,
  M5, M6, M7, DECODER1
  imports:
    IN2 SIGNAL BOOLEAN AND-M7-B ( OUT1 DRIVER SWITCH-IN-X
      ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
      IN1 SIGNAL BOOLEAN AND-M7-B ( OUT1 DECODER1 AND-M7-A
        ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
      IN2 SIGNAL BOOLEAN AND-M7-A ( OUT1 DRIVER SWITCH-IN-Y
        ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
      IN1 SIGNAL BOOLEAN AND-M7-A ( OUT1 DRIVER SWITCH-IN-Z
        ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
      IN2 SIGNAL BOOLEAN AND-M6-B ( OUT1 DRIVER SWITCH-IN-X
        ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
```

```
IN1 SIGNAL BOOLEAN AND-M6-B ( OUT1 DECODER1 AND-M6-A
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M6-A ( OUT1 DRIVER SWITCH-IN-Y
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M6-A ( OUT1 DECODER1 NOT-Z
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M5-B ( OUT1 DRIVER SWITCH-IN-X
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M5-B ( OUT1 DECODER1 AND-M5-A
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M5-A ( OUT1 DECODER1 NOT-Y
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M5-A ( OUT1 DRIVER SWITCH-IN-Z
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M4-B ( OUT1 DRIVER SWITCH-IN-X
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M4-B ( OUT1 DECODER1 AND-M4-A
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M4-A ( OUT1 DECODER1 NOT-Y
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M4-A ( OUT1 DECODER1 NOT-Z
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M3-B ( OUT1 DECODER1 NOT-X
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M3-B ( OUT1 DECODER1 AND-M3-A
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M3-A ( OUT1 DRIVER SWITCH-IN-Y
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M3-A ( OUT1 DRIVER SWITCH-IN-Z
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M2-B ( OUT1 DECODER1 NOT-X
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M2-B ( OUT1 DECODER1 AND-M2-A
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M2-A ( OUT1 DRIVER SWITCH-IN-Y
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M2-A ( OUT1 DECODER1 NOT-Z
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M1-B ( OUT1 DECODER1 NOT-X
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M1-B ( OUT1 DECODER1 AND-M1-A
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M1-A ( OUT1 DECODER1 NOT-Y
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M1-A ( OUT1 DRIVER SWITCH-IN-Z
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M0-B ( OUT1 DECODER1 NOT-X
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN1 SIGNAL BOOLEAN AND-M0-B ( OUT1 DECODER1 AND-M0-A
 ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
IN2 SIGNAL BOOLEAN AND-M0-A ( OUT1 DECODER1 NOT-Y
      ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
    IN1 SIGNAL BOOLEAN AND-M0-A ( OUT1 DECODER1 NOT-Z
      ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
    IN1 SIGNAL BOOLEAN NOT-Z ( OUT1 DRIVER SWITCH-IN-Z
      ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
    IN1 SIGNAL BOOLEAN NOT-Y ( OUT1 DRIVER SWITCH-IN-Y
      ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
    IN1 SIGNAL BOOLEAN NOT-X ( OUT1 DRIVER SWITCH-IN-X
      ) import-path: ( DRIVER DECODER1 ) import-owner: DECODER1
    IN1 SIGNAL BOOLEAN M7 ( OUT1 DECODER1 AND-M7-B
      ) import-path: ( DRIVER ) import-owner: DRIVER
    IN1 SIGNAL BOOLEAN M6 ( OUT1 DECODER1 AND-M6-B
      ) import-path: ( DRIVER ) import-owner: DRIVER
    IN1 SIGNAL BOOLEAN M5 ( OUT1 DECODER1 AND-M5-B
      ) import-path: ( DRIVER ) import-owner: DRIVER
    IN1 SIGNAL BOOLEAN M4 ( OUT1 DECODER1 AND-M4-B
      ) import-path: ( DRIVER ) import-owner: DRIVER
    IN1 SIGNAL BOOLEAN M3 ( OUT1 DECODER1 AND-M3-B
      ) import-path: ( DRIVER ) import-owner: DRIVER
    IN1 SIGNAL BOOLEAN M2 ( OUT1 DECODER1 AND-M2-B
      ) import-path: ( DRIVER ) import-owner: DRIVER
    IN1 SIGNAL BOOLEAN M1 ( OUT1 DECODER1 AND-M1-B
      ) import-path: ( DRIVER ) import-owner: DRIVER
    IN1 SIGNAL BOOLEAN M0 ( OUT1 DECODER1 AND-M0-B
      ) import-path: ( DRIVER ) import-owner: DRIVER
exports:
  OUT1 SIGNAL BOOLEAN AND-M7-B
    export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M7-A
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M6-B
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M6-A
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M5-B
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M5-A
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M4-B
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M4-A
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M3-B
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M3-A
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M2-B
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M2-A
      export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
    OUT1 SIGNAL BOOLEAN AND-M1-B
```

```
        export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
      OUT1 SIGNAL BOOLEAN AND-M1-A
        export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
      OUT1 SIGNAL BOOLEAN AND-M0-B
        export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
      OUT1 SIGNAL BOOLEAN AND-M0-A
        export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
      OUT1 SIGNAL BOOLEAN NOT-Z
        export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
      OUT1 SIGNAL BOOLEAN NOT-Y
        export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
      OUT1 SIGNAL BOOLEAN NOT-X
        export-path: ( DRIVER DECODER1 ) export-owner: DECODER1
      OUT1 SIGNAL BOOLEAN SWITCH-IN-Z
        export-path: ( DRIVER ) export-owner: DRIVER
      OUT1 SIGNAL BOOLEAN SWITCH-IN-Y
        export-path: ( DRIVER ) export-owner: DRIVER
      OUT1 SIGNAL BOOLEAN SWITCH-IN-X
        export-path: ( DRIVER ) export-owner: DRIVER
  initialize procedure: update procedure:
  update SWITCH-IN-X update SWITCH-IN-Y update SWITCH-IN-Z
    update DECODER1 update M0 update M1 update M2 update M3
    update M4 update M5 update M6 update M7
not-gate NOT-X delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 600 comp-y: 650
not-gate NOT-Y delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 200 comp-y: 650
not-gate NOT-Z delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 200 comp-y: 250
and-gate AND-M0-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 400 comp-y: 800
and-gate AND-M0-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 800 comp-y: 800
and-gate AND-M1-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 400 comp-y: 700
and-gate AND-M1-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 800 comp-y: 700
and-gate AND-M2-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 400 comp-y: 600
and-gate AND-M2-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 800 comp-y: 600
and-gate AND-M3-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 400 comp-y: 500
and-gate AND-M3-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 800 comp-y: 500
and-gate AND-M4-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 400 comp-y: 400
and-gate AND-M4-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 800 comp-y: 400
and-gate AND-M5-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 400 comp-y: 300
```

```
and-gate AND-M5-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 800 comp-y: 300
and-gate AND-M6-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 400 comp-y: 200
and-gate AND-M6-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 800 comp-y: 200
and-gate AND-M7-A delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 400 comp-y: 100
and-gate AND-M7-B delay: 0 not mil-spec manufacturer: " "
  power level: 0.0 comp-x: 800 comp-y: 100
subsystem DECODER1 is controls:
  NOT-X, NOT-Y, NOT-Z, AND-M0-A, AND-M0-B, AND-M1-A,
  AND-M1-B, AND-M2-A, AND-M2-B, AND-M3-A, AND-M3-B, AND-M4-A,
  AND-M4-B, AND-M5-A, AND-M5-B, AND-M6-A, AND-M6-B, AND-M7-A,
  AND-M7-B
  imports: exports: initialize procedure: update procedure:
  update NOT-X update NOT-Y update NOT-Z update AND-M0-A
    update AND-M0-B update AND-M1-A update AND-M1-B
    update AND-M2-A update AND-M2-B update AND-M3-A
    update AND-M3-B update AND-M4-A update AND-M4-B
    update AND-M5-A update AND-M5-B update AND-M6-A
    update AND-M6-B update AND-M7-A update AND-M7-B
  comp-x: 500 comp-y: 200
```

Appendix D. Sample Session : Populating OODBMS With Domain Knowledge

This appendix contains a sample session for using the ITASCA Active Data Editor to populate the Domain-Definition Meta-Model with a subset of the logic circuits domain. The portion of the logic circuits domain implemented in this appendix is shaded and labeled "sample session" in Figure D.1. The rest of the logic circuits domain was implemented in the same way, but is not shown in this appendix.

Figure D.2 is an instance diagram of the Domain-Definition Meta-Model for the subset of the logic circuits domain we instantiate in this appendix. We have found it best to start at the bottom of the instance diagram and work toward the top because the lower-level objects generally need to be put into some relationship attribute of a higher-level object. For instance, you must instantiate the DATA-OBJECT classes in the lower right hand corner of Figure D.2 before you can completely define the CONCRETE class above it. Those DATA-OBJECT instances are identified by their unique object identifier in their parent CONCRETE classes ICO-DATA-OBJECTS attribute.

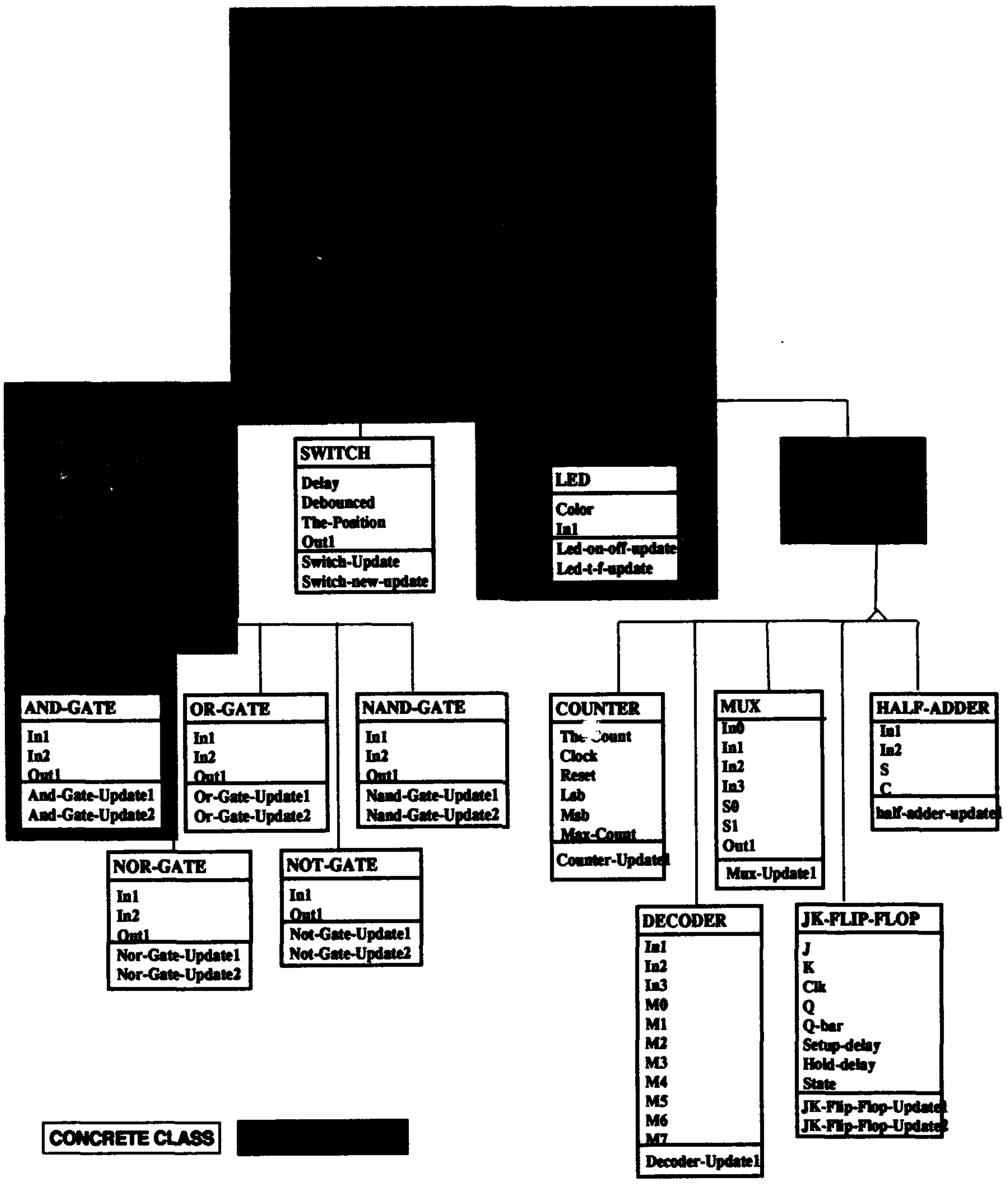
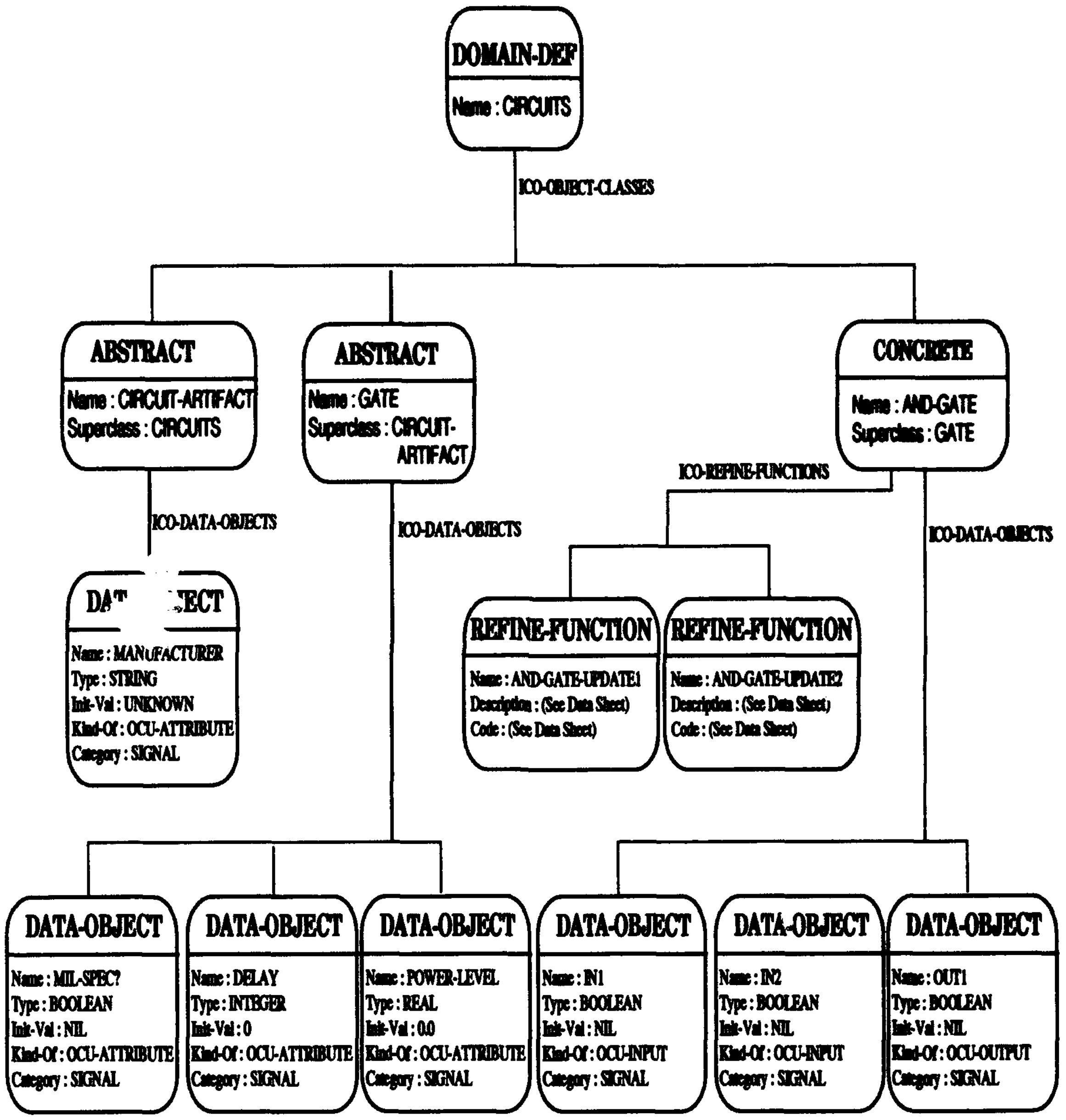


Figure D.1 Sample Subset of Logic Circuits Domain



Func-Type-Enum (OCU-Active-Update, OCU-Update)
Kind-Enum (OCU-Attribute,OCU- Constant, OCU-Coefficient, OCU-Input, OCU-Output)

Figure D.2 Instance Diagram for Sample Subset of Logic Circuits Domain

D.1 Start a Session

When the ITASCA ADE is invoked, the window depicted in Figure D.3 is displayed. The first thing to do after invoking the ADE is to move to the appropriate database in which to do your work. This is done by clicking on the "Private" icon on the menu bar and selecting from the list of databases that will be displayed in a sub-menu. As you can see from the message on the window, we are currently connected to private database thirteen to do this work. Once you are in the appropriate database, you should commit the "move" transaction by first clicking on "ITASCA" and then clicking on "Commit" in the sub-menu that comes up.

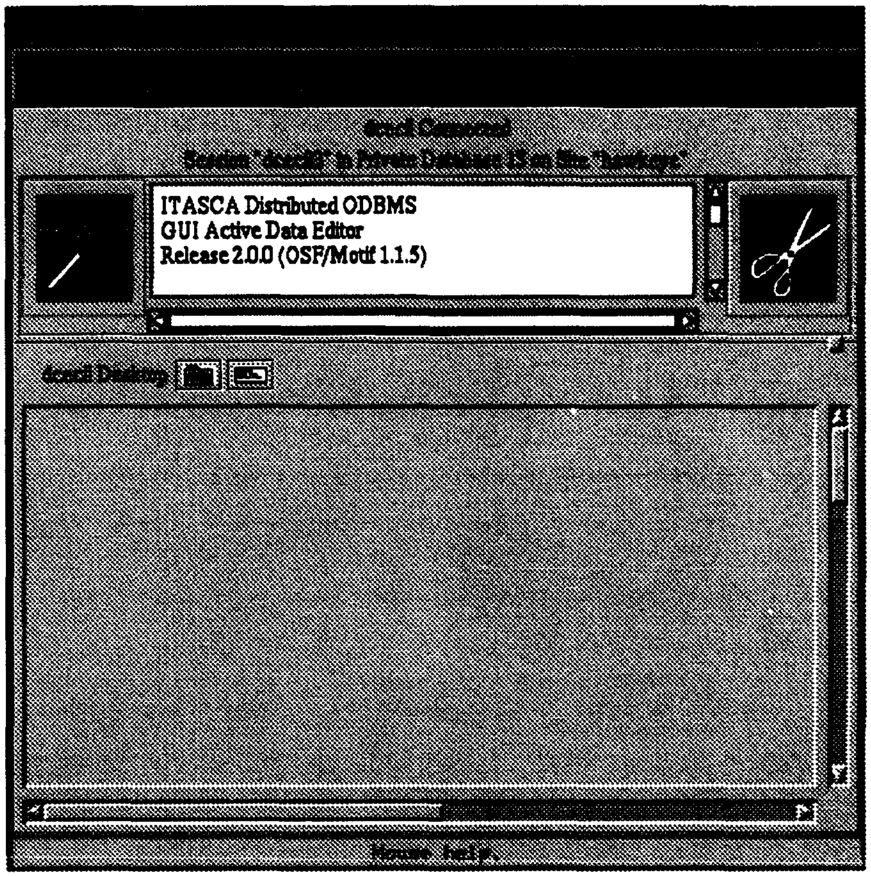
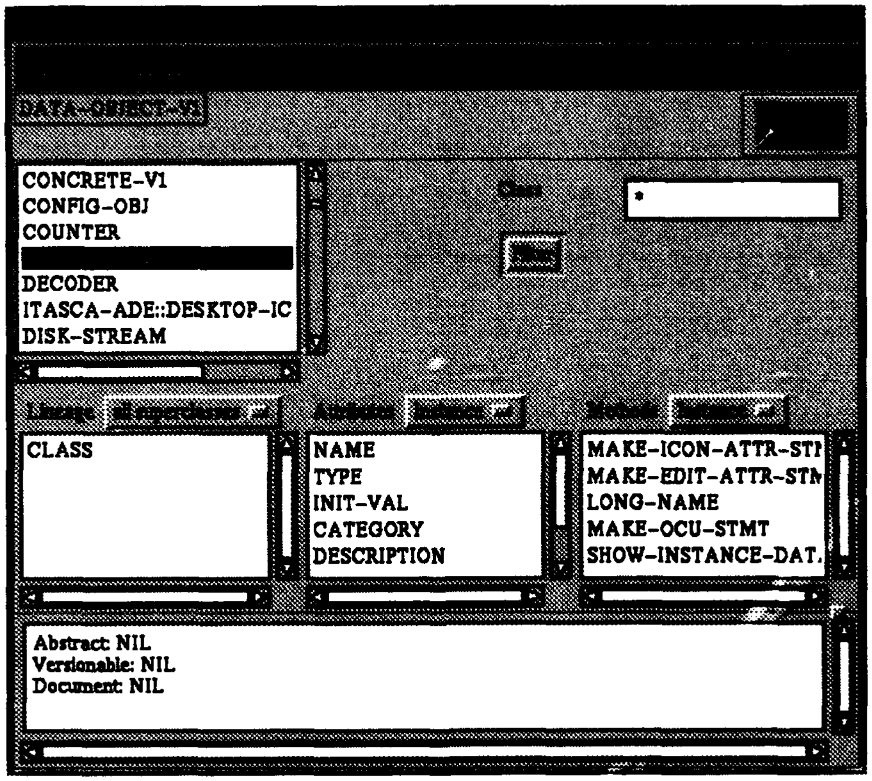

ITASCA Distributed ODBMS
GUI Active Data Editor
Release 2.0.0 (OSF/Motif 1.1.5)

Figure D.3 ITASCA ADE

D.2 Instantiate Instances of the DATA-OBJECT Class

We start by instantiating all the DATA-OBJECT classes (attributes, inputs, and outputs) shown in Figure D.2. Click on "View" in Figure D.3, then when the sub-menu appears click on "Schema" to cause the window in Figure D.4 to appear. Click on "Filter" in Figure D.4 to get the classes in the window to the left of the "Filter" to display, and select "DATA-OBJECT-V1" as illustrated. Click on "Object" in the menu bar, and "New Instance" in the sub-menu to bring up a blank template for defining an instance of DATA-OBJECT Class. Figure D.5 is a New Instance template filled in for the manufacturer attribute depicted in Figure D.1 and Figure D.2. We go through this same process for the rest of the DATA-OBJECT instances depicted in Figure D.2. Figure D.6 through Figure D.11 illustrate the creation of the rest of the (attribute, input, and output) DATA-OBJECT instances.



CONCRETE-V1
CONFIG-OBJ
COUNTER
DECODER
ITASCA-ADE::DESKTOP-IC
DISK-STREAM

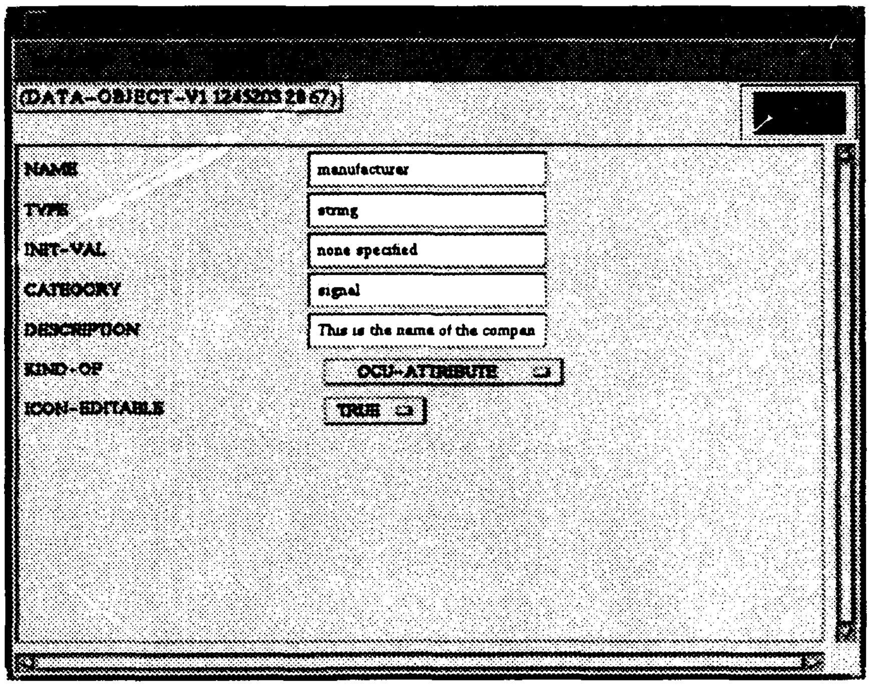
CLASS

NAME
TYPE
INIT-VAL
CATEGORY
DESCRIPTION

MAKE-ICON-ATTR-STI
MAKE-EDIT-ATTR-STM
LONG-NAME
MAKE-OCU-STMT
SHOW-INSTANCE-DAT.

Abstract: NIL
Versionable: NIL
Document: NIL

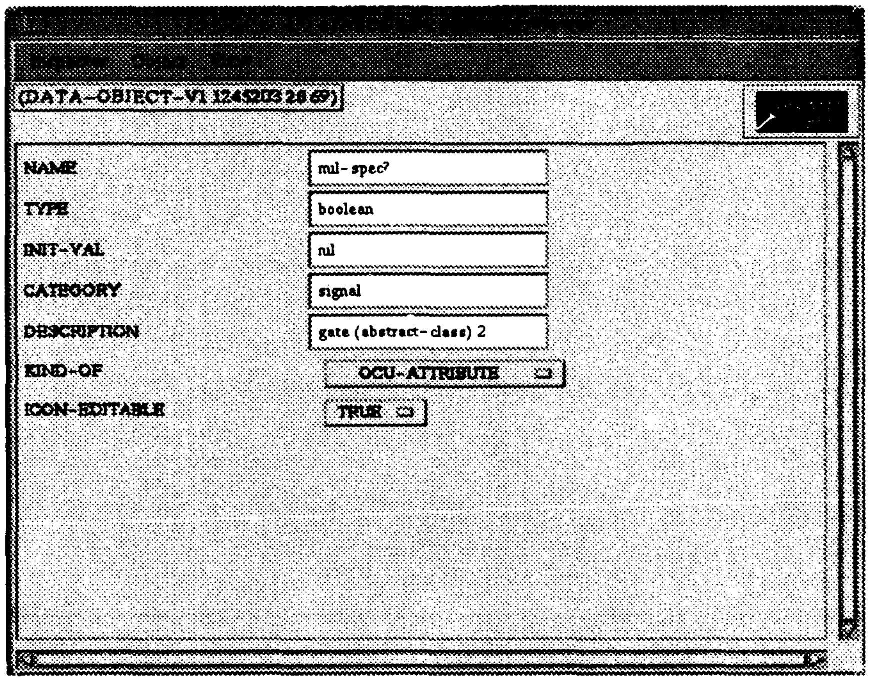
Figure D.4 Selecting the DATA-OBJECT Class



(DATA-OBJECT-V1 12452033 28 67)

NAME	manufacturer
TYPE	string
INIT-VAL	none specified
CATEGORY	signal
DESCRIPTION	This is the name of the compan
KIND-OF	OCU-ATTRIBUTE
ICON-EDITABLE	TRUE

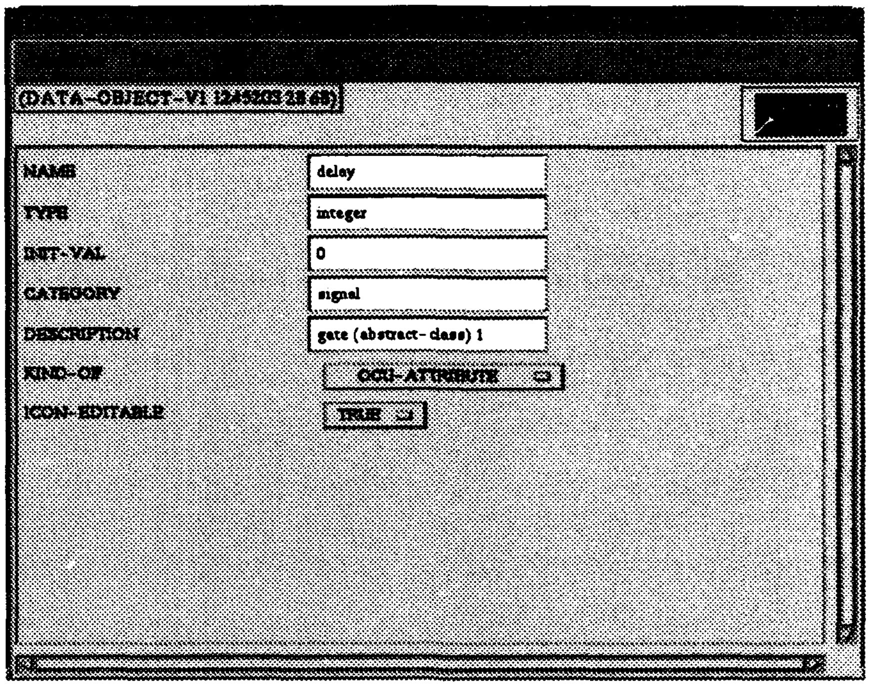
Figure D.5 Instantiating the Manufacturer Attribute



(DATA-OBJECT-V1 12452033 28 69)

NAME	mil-spec?
TYPE	boolean
INIT-VAL	nil
CATEGORY	signal
DESCRIPTION	gate (abstract-class) 2
KIND-OF	OCU-ATTRIBUTE
ICON-EDITABLE	TRUE

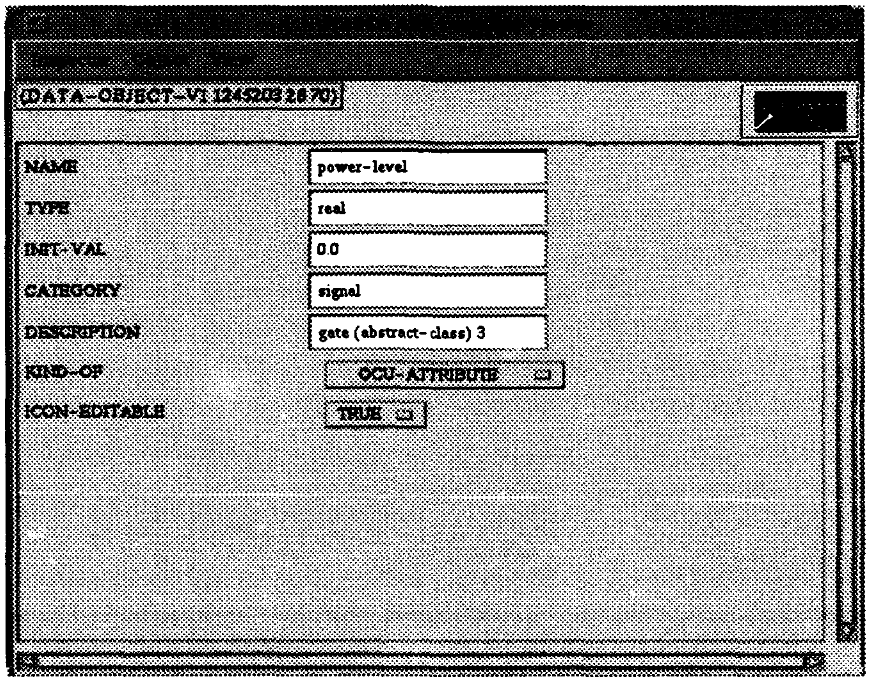
Figure D.6 Instantiating the MilSpec? Attribute



(DATA-OBJECT-V1 1245203 28 68)

NAME	delay
TYPE	integer
INIT-VAL	0
CATEGORY	signal
DESCRIPTION	gate (abstract-class) 1
KIND-OF	OCU-ATTRIBUTE
ICON-EDITABLE	TRUE

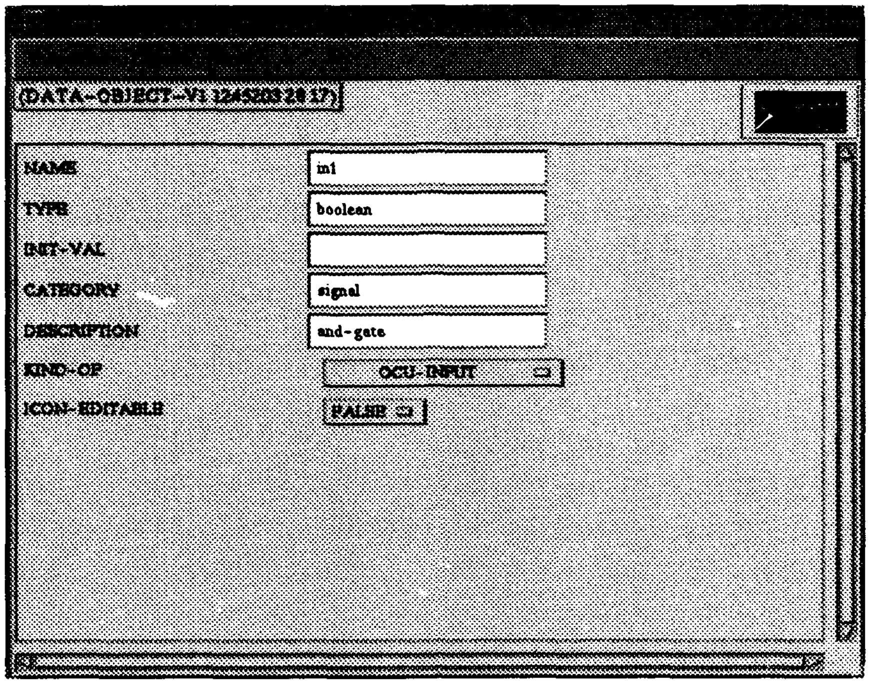
Figure D.7 Instantiating the Delay Attribute



(DATA-OBJECT-V1 1245203 28 70)

NAME	power-level
TYPE	real
INIT-VAL	0.0
CATEGORY	signal
DESCRIPTION	gate (abstract-class) 3
KIND-OF	OCU-ATTRIBUTE
ICON-EDITABLE	TRUE

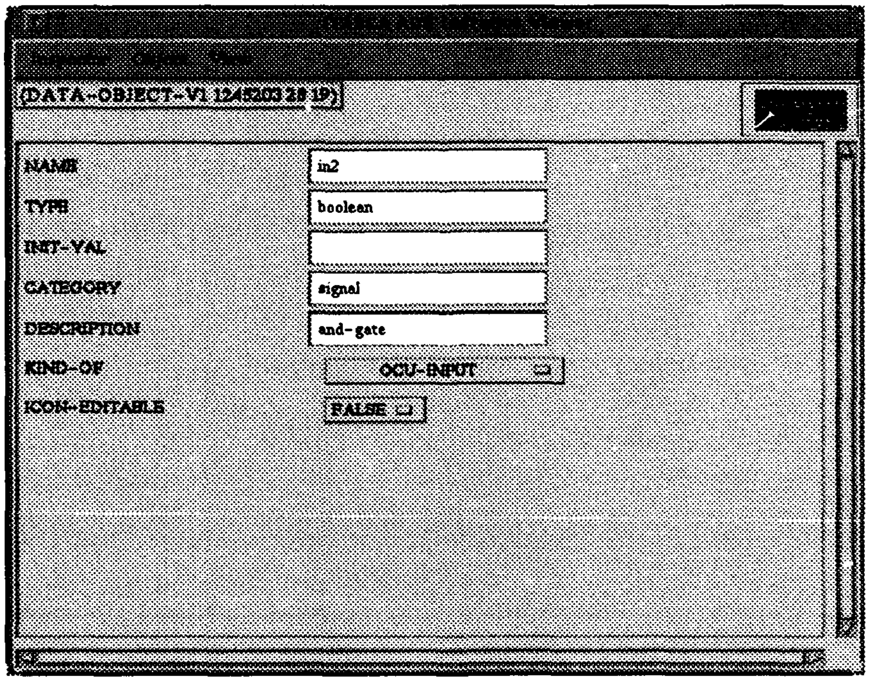
Figure D.8 Instantiating the PowerLevel Attribute



(DATA-OBJECT-V1 1245203 28 17)

NAME	in1
TYPE	boolean
INIT-VAL	
CATEGORY	signal
DESCRIPTION	and-gate
KIND-OF	OCU-INPUT
ICON-EDITABLE	FALSE

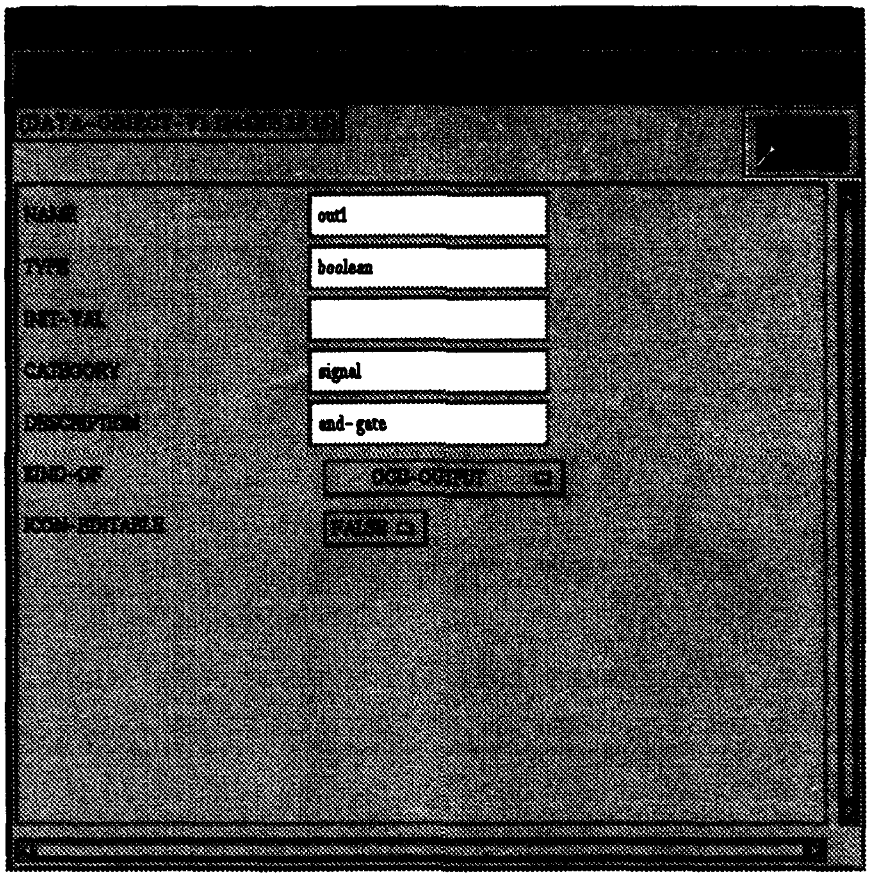
Figure D.9 Instantiating the In1 Input



(DATA-OBJECT-V1 1245203 28 19)

NAME	in2
TYPE	boolean
INIT-VAL	
CATEGORY	signal
DESCRIPTION	and-gate
KIND-OF	OCU-INPUT
ICON-EDITABLE	FALSE

Figure D.10 Instantiating the In2 Input

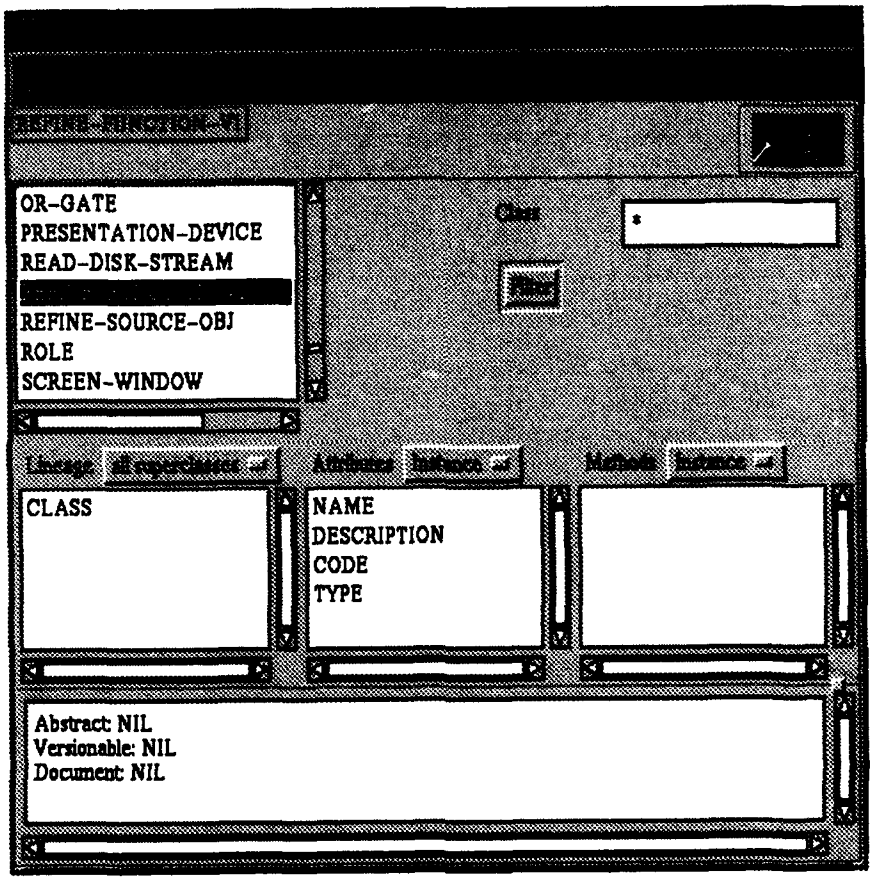


NAME	out1
TYPE	boolean
INIT-VAL	
CATEGORY	signal
DESCRIPTION	and-gate
KIND-OF	DCB-OUTPUT
ICON-EDITABLE	FALSE

Figure D.11 Instantiating the Out1 Output

D.3 Instantiate Instances of the REFINE-FUNCTION Class

Next we instantiate the two REFINE-FUNCTION instances depicted in Figure D.2. Select the REFINE-FUNCTION class as illustrated in Figure D.12. As before, click on "Object" and "New Instance" to get a template for instantiating the REFINE-FUNCTION class. Figure D.13 and Figure D.14 are the two instantiations for the AND-GATE-UPDATE1 and AND-GATE-UPDATE2 functions respectively.



Abstract: NIL
Versionable: NIL
Document: NIL

Figure D.12 Selecting the REFINE-FUNCTION Class

THIS

PAGE

IS

MISSING

IN

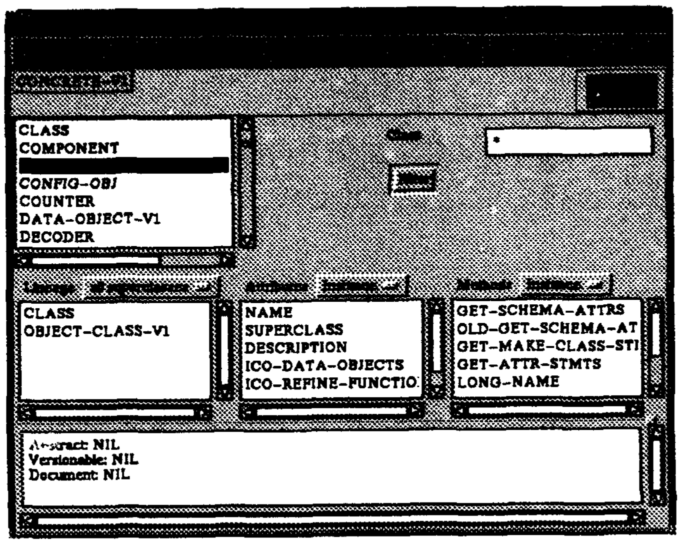
ORIGINAL

DOCUMENT

Figures D.13 thru D.17

D.5 Instantiate Instances of the CONCRETE Class

Now instantiate the only CONCRETE class for this example depicted in Figure D.2. Select the CONCRETE class as illustrated in Figure D.18. Click on "Object" and "New Instance" to get a template for instantiating the CONCRETE class. Figure D.19 is the instantiation for the And-Gate CONCRETE class.



```
CLASS
COMPONENT

CONFIG-OBJ
COUNTER
DATA-OBJECT-V1
DECODER
```

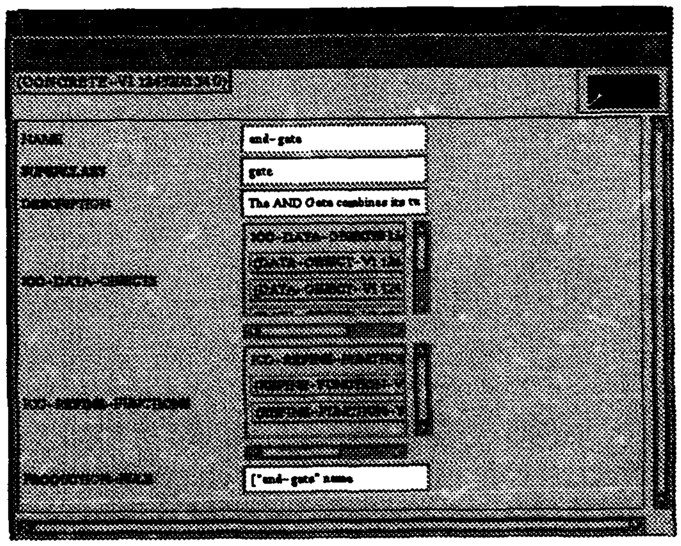
```
CLASS
OBJECT-CLASS-V1
```

```
NAME
SUPERCLASS
DESCRIPTION
ICO-DATA-OBJECTS
ICO-REFINE-FUNCTIO
```

```
GET-SCHEMA-ATTRS
OLD-GET-SCHEMA-AT
GET-MAKE-CLASS-STI
GET-ATTR-STMTS
LONG-NAME
```

```
Abstract: NIL
Versionable: NIL
Document: NIL
```

Figure D.18 Selecting the CONCRETE Class



NAME	and-gate
SUPERCLASS	gate
DESCRIPTION	The AND Gate combines its tw
ICO-DATA-OBJECTS	[illegible]
ICO-REFINE-FUNCTIONS	[illegible]
[illegible]	["and-gate" name

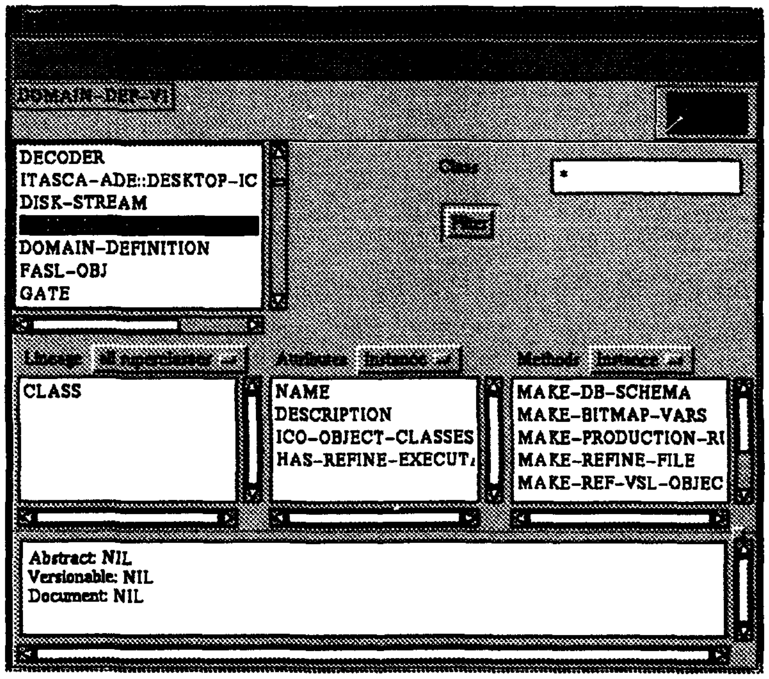
Figure D.19 Instantiating the CONCRETE Class for And-Gate

D.6 Instantiate Instances of the DOMAIN-DEF Class

Finally, instantiate the DOMAIN-DEF class depicted in Figure D.2. Select the DOMAIN-DEF class as shown in Figure D.20. Click on "Object" and "New Instance" to get a template for instantiating the DOMAIN-DEF class. Figure D.21 is the instantiation for the logic circuits DOMAIN-DEF class.

Notice in Figure D.21 that inside the window labeled ICO-OBJECT-CLASSES there are a number of objects with labels on them. These objects correspond to ABSTRACT and CONCRETE objects we instantiated earlier in this session. They are put in the ICO-OBJECT-CLASSES attribute by clicking on an objects label and dragging and dropping it into this window. The same thing is done for the ICO-DATA-OBJECTS and ICO-REFINE-FUNCTIONS relationships like those pictured in Figure D.19.

This completes the process of loading the definition of a domain into the database. At this poinı the domain knowledge is available for manipulation and transformation.



```
DECODER
ITASCA-ADE::DESKTOP-IC
DISK-STREAM

DOMAIN-DEFINITION
FASL-OBJ
GATE
```

```
*
```

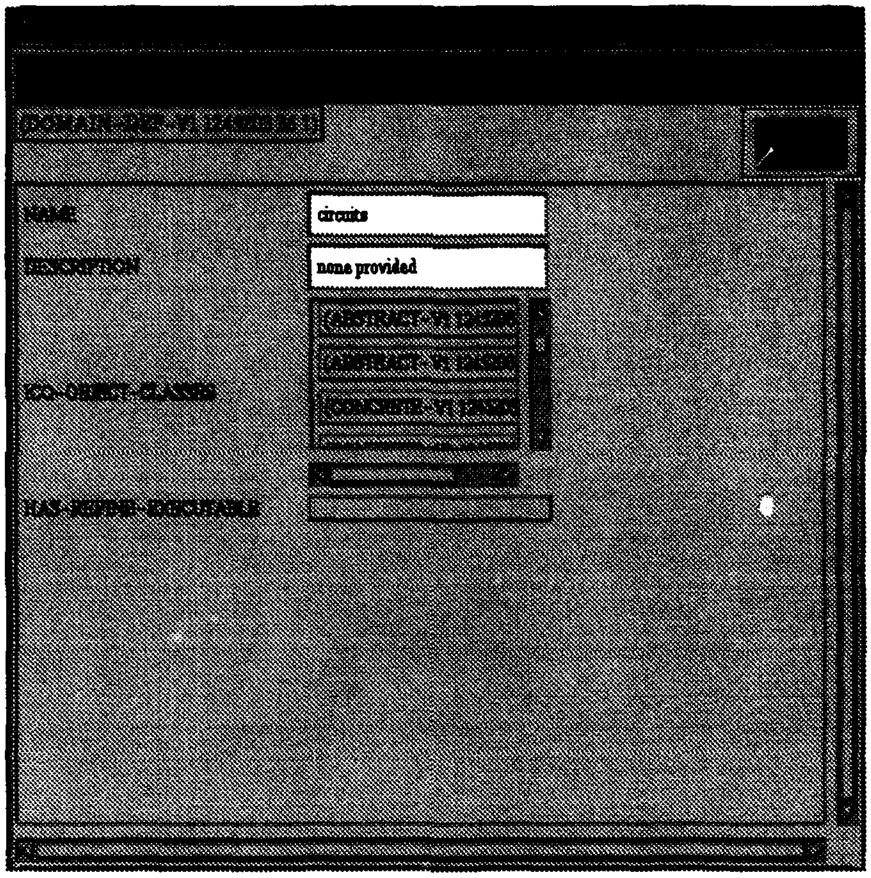
```
CLASS
```

```
NAME
DESCRIPTION
ICO-OBJECT-CLASSES
HAS-REFINE-EXECUT
```

```
MAKE-DB-SCHEMA
MAKE-BITMAP-VARS
MAKE-PRODUCTION-RI
MAKE-REFINE-FILE
MAKE-REF-VSL-OBJEC
```

Abstract: NIL
Versionable: NIL
Document: NIL

Figure D.20 Selecting the DOMAIN-DEF Class



(DOMAIN-DEF-V1 [illegible])

NAME circuits

DESCRIPTION none provided

ICO-OBJECT-CLASSES (ABSTRACT-V1 [illegible]
(ABSTRACT-V1 [illegible]
(CONCRETE-V1 [illegible]

HAS-REFINE-EXECUTABLE

Figure D.21 Instantiating the DOMAIN-DEF Class for Circuits Domain

Appendix E. Sample Session : OODBMS Implementation of Technology Base

This appendix contains a sample session in which a simple application for a light is built from the primitive objects defined in the logic-circuits domain, and saved to the database. The circuit diagram for the Light is shown in Figure E.1.

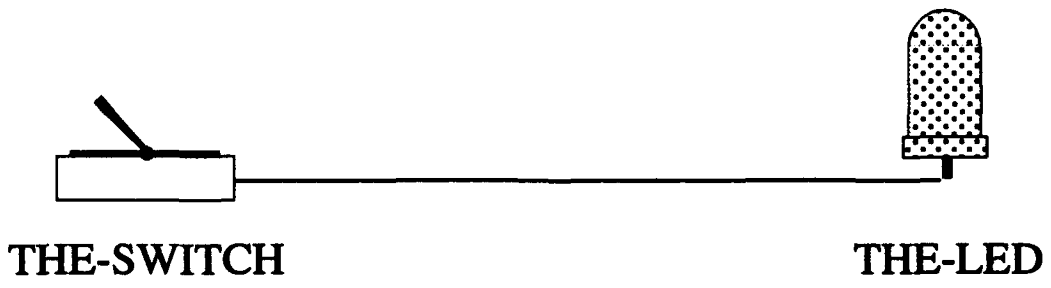


Figure E.1 Light Circuit

E.1 Start AVSI

REFINE must be loaded in an Emacs window. Once this is accomplished enter:

```
(load "dbl")
```

When the prompt returns, enter:

```
(dbl)
```

It will take several minutes for this file to run because it loads the DIALECT and INTERVISTA systems from the UNIX file system, and loads the Architect and AVSI files from the database. When the load is complete, the following prompt appears:

```
Load Complete
Type "(AVSI)" to start AVSI
```

Now enter the command:

```
(avsi)
```

This action loads the visual specification files for the domains currently defined for Architect. After the visual information is parsed into the object base, the control panel appears in the upper left-hand corner of the screen. Across the top of the window is a row of icons that will be used to invoke many of the application composition functions of AVSI. The lower portion of the window is a message area used by AVSI to display status and error messages.

E.2 Create a New Application

1. Click any mouse button on the icon labeled CREATE NEW APPLICATION.
2. A pop-up window appears and prompts, SELECT DOMAIN. Click on the menu item CIRCUITS. At this time, the Circuits domain definition is loaded from the database, as are the bitmap images for each primitive.
3. A pop-up window appears with the prompt, ENTER NAME OF APPLICATION. Type

   `light`

4. The name can be entered by hitting the "return" key or by clicking on DO IT at the bottom of the pop-up window.

E.3 Edit the Application

Now that the application has been created, the next step is to edit the application's make-up. Editing an application is comprised of two separate operations: editing an application's components, and editing an application's update algorithm.

E.3.1 Add the Controlling Subsystem-obj to the Application.

1. Click a mouse button on the EDIT APPLICATION control panel icon.
2. A pop-up menu appears with the prompt CHOOSE APPLICATION. Click on the menu item LIGHT.

3. A pop-up menu appears with the prompt Choose. Click on the menu item Edit Application Components. A blue window appears containing a single icon labeled,
   APPLICATION – OBJ
   LIGHT .

4. Click on the diagram surface (anywhere on the blue surface except within the icon's boundary) of the window. A pop-up menu will appear.

5. Select Create New Subsystem.

6. A pop-up window appears, with the prompt Enter a name. Enter `the-light`

7. A box outline of an icon appears, attached to the mouse cursor. Place the icon below the application-obj icon by moving the cursor to the desired location and clicking.

8. Click any mouse button on the newly created subsystem-obj icon and select the menu option Link to Source.

9. The mouse cursor changes from an arrow to an oval with a dot in it, signifying that an object needs to be selected. Place the mouse cursor on the application-obj's icon and click any mouse button. A link appears between the application-obj's icon and the subsystem-obj's icon.

10. Close the edit-application-window by clicking on the diagram surface and selecting Deactivate.

E.3.2 Create the Application-obj's Update Algorithm.

1. Click a mouse button on the Edit Application control panel icon.

2. A pop-up menu appears with the prompt Choose Application. Click on the menu item LIGHT.

3. A pop-up menu appears with the prompt Choose. Click on the menu item Edit Application Update. Three windows will appear. One contains a graphical view of the update algorithm, one contains a textual view of the algorithm, and the third (the Controllee Window) shows the icons that represent the application-obj's controllees (with two extra icons for if-then and while-do constructs). The graphical update

window contains two icons, "Start" and "End", with a dotted arrow pointing from the start-icon to the end-icon.

4. Click a mouse button on the icon in the controllee window labeled SUBSYSTEM - OBJ THE - LIGHT. The cursor changes to an oval with a dot in it indicating that an object needs to be selected.
5. Click on the "nub" on the dotted line midway between the start and end icons. This will cause the update sequence to redraw with the subsystem-obj included. Note the textual representation is automatically updated to reflect each change in the diagram window.
6. Close the edit-update-algorithm windows by clicking on the black title bar at the top of the graphical update window and selecting DEACTIVATE.

E.4 Edit the Subsystems

Building a subsystem is similar to building the application. This section illustrates how to instantiate primitive-objs and nested subsystem-obj's and link them to the controlling subsystem created in the previous section.

The subsystem, THE-LIGHT, will control a switch and an LED. To add these objects, perform the following steps:

E.4.1 Add the Subsystem.

1. Click on the EDIT SUBSYSTEM icon in the control panel window.
2. Click on the menu item THE-LIGHT. A white window opens (the subsystem window) which contains an OCU representation of a subsystem.
3. Click on the OBJECTS icon in the subsystem window. The blue edit-subsystem-window for THE-LIGHT appears, containing a single icon labeled SUBSYSTEM - OBJ THE - LIGHT. A green window, the technology-base window, also appears and contains an icon for each primitive-object in the current domain.

E.4.2 Add the Primitive Objects.

1. Click on the icon in the green technology-base-window labeled SWITCH-OBJ, that looks like a toggle switch.
2. A Switch-icon is created and attached to the mouse cursor. Place this icon on the blue edit-subsystem-window near THE-LIGHT.
3. Name the switch by typing THE-SWITCH in the pop-up window.
4. Similarily, create an LED object named THE-LED.
5. Link the primitive objects to THE-LIGHT by clicking a mouse button on THE-LIGHT, and selecting LINK MULTIPLE TARGETS from the pop-up menu.
6. A pop-up window will appear that lists all the unconnected objects in the edit-subsystem-window. Select ALL OF THE ABOVE, and click on DO IT. A link will appear from THE-LIGHT to each of the other icons.
7. Close the edit-subsystem-window and the technology-base-window by clicking on the blue surface and selecting DEACTIVATE. The windows can be closed separately by selecting DEACTIVATE from their title bar menus.

   At this point, the subsystem window for THE-LIGHT will again be visible.

E.4.3 Connect LIGHT's Imports and Exports. To connect the import and export objects perform the following steps:

1. Click a mouse button on the IMPORT AREA or EXPORT AREA icon in LIGHT's subsystem window.
2. Select MAKE CONNECTIONS from the pop-up window. A red window (the import-export-window) will open and contain the switch icon and the led icon. The black bars (these bars are actually highlighted subicons attached to the primitive's icon) on the sides of the primitive icons indicate connections that need to be made.
3. Icons can be moved to new positions on the screen by clicking on the icon and selecting MOVE ICON from the pop-up menu. A square grid, attached to the mouse,

appears. Move the grid to the new icon position and click to "drop" the icon. Move the LED icon to the right of the SWITCH icon so the black bars are facing each other.

4. Connect switch THE-SWITCH to led THE-LED by clicking on the black bar of THE-SWITCH and then clicking on the black bar of THE-LED.
5. Close the import-export-window and the msp-windows by clicking on the red surface and selecting DEACTIVATE, or by selecting DEACTIVATE from each window's title bar menu.

E.4.4 Build THE-LIGHT*'s Update Algorithm.* After the import-export windows have been closed, the white subsystem window will again be visible. Building the update algorithm for THE-LIGHT is similar to building the update algorithm for the application, and requires the following steps:

1. Click the mouse on the CONTROLLER icon in the subsystem window. The three windows that were seen before are exposed, except the controllee window now contains the switch and led controlled by THE-LIGHT.
2. Add each controllee to the update sequence by clicking on the controllee icon and then clicking on the "nub" in the graphical update window that represents the proper sequence position for the controllee. The order in which the controllees must appear is:

   ```
   THE-SWITCH      THE-LED
   ```

   Note that the textual update description is updated as the graphical update is built.
3. Close the windows by clicking on the graphical update window title bar and selecting DEACTIVATE.

E.5 Perform Semantic Checks

Semantic checks are performed by Architect as part of the import-export connection process. However, the semantic checks may be run at any time by clicking on the control panel icon labeled CHECK SEMANTICS. The results of the semantic checks may be viewed in the EMACS window.

E.6 Execute the Application

Now that the application has been fully defined, it can be executed. The default position for a switch is "on", however the switch attributes can be changed for each execution. To set the switch attributes, do the following:

1. Click on the control panel's EDIT SUBSYSTEM icon.
2. Choose THE-LIGHT from the pop-up window.
3. Click on the OBJECTS icon in the subsystem window.
4. Click on the switch icon labeled THE-SWITCH.
5. Choose VIEW/EDIT ATTRIBUTES from the pop-up window. A window appears, listing the switch attributes for THE-SWITCH.
6. Click on the attribute, POSITION. A pop-up window appears, listing the current value of the switch.
7. Enter a new value for the switch position by typing

   ```
   avsi::off
   ```

   (The "avsi::" prefix is the package name and is required for symbols. It is not required for other data types such as numbers and strings)

Once the input state has been achieved, click on the control panel button labeled EXECUTE APPLICATION. The results are displayed in the EMACS window. A new switch position can be established, and the application can be re-executed.

E.7 Save the Application to the Database

Now that the application has been successfully composed, it can be saved to the database. To save the application, do the following:

1. Click on the control panel's EDIT APPLICATION icon.

2. A pop-up menu appears with the prompt CHOOSE APPLICATION. Click on the menu item LIGHT.

3. A pop-up menu appears with the prompt CHOOSE. Click on the menu item SAVE APPLICATION TO DATABASE. The application will now be copied to the database. The debug information for the database transactions should appear in the EMACS window.

First, an OCU-APPLICATION object will be created, and its NAME, DOMAIN, and APPLICATION-MODE attributes will be set (refer to E.2).

```
(apply (function MAKE) (quote (OCU-APPLICATION)))

(apply (function SETF-SEND) (quote (NAME #(851981 346 57) NIL LIGHT)))

(apply (function SETF-SEND) (quote (DOMAIN #(851981 346 57) NIL CIRCUITS)))

(apply (function SETF-SEND) (quote (APPLICATION-MODE #(851981 346 57) NIL
NON-EVENT-DRIVEN-SEQUENTIAL)))
```

Figure E.2 OCU-APPLICATION Transaction

Similarly, the OCU-SUBSYSTEM, LED, and SWITCH object are created. An OCU-UPDATE-STATEMENT is created with the OCU-SUBSYSTEM as its TARGET association and is placed in the OCU-APPLICATION UPDATE-FUNCTION association. Two more OCU-UPDATE-STATEMENT objects are created with the LED and SWITCH as their TARGETs and are placed in the OCU-SUBSYSTEM UPDATE-FUNCTION association. An OCU-OUTPUT is created for the SWITCH and its attributes are updated, and an OCU-INPUT is created and updated for the LED. Once all of the objects have been created, their attributes set, and their associations defined, a database COMMIT is sent. This signifies the end of the database transaction, and at this time the objects are saved in the database.

E.8 Delete the Application from the Database

Since this was merely an exercise to show how the database functionality is transparent to the AVSI user, we will now delete the application from the database. This will allow other users to perform this exercise and write to the database. In order to delete an application from the database, do the following:

1. Click any mouse button on the icon labeled SAVED APPLICATION UTILITIES.
2. A pop-up window appears and prompts, ENTER CHOICE. Click on the menu item REMOVE APPLICATION FROM DATABASE.
3. A pop-up window appears and prompts, CHOOSE DOMAIN. Click on the menu item CIRCUITS.
4. A pop-up window appears with a list of applications currently stored in the database for the Circuits domain. Click on the menu item LIGHT. At this time, the application will be removed from the database. Again, the database transactions will scroll on the EMACS window. After each object contained within the application has been deleted, a COMMIT is sent signalling the end of the database transaction. At this time, the objects are no longer accessible in the database.

Bibliography

1. Anderson, Cynthia. *Creating and Manipulating Formalized Software Architectures in Support of a Domain-Oriented Application Composition System.* MS thesis, AFIT/GCS/ENG/92D-01, Air Force Institute of Technology, December 1992.

2. ASC/RWWW. *System Concept Document for the Joint Modeling and Simulation System (J-MASS) Program.* CROSSBOW-S Architecture Technical Working Group, November 1991.

3. Atkinson, Malcom, et al. "The Object-Oriented Database System Manifesto." *Proceedings of the First International Conference on Deductive and Object-Oriented Databases.* December 1989.

4. Bailor, Major Paul D., "CSCE 793 - Formal-Based Methods in Software Engineering." Class Notes, 1993.

5. Bailor, Paul D. and others. "Formalization and Visualization of Domain-Specific Software Architectures," *AAAI-92 Workshop on Automated Software Design, AAAI Conference*, 6–11 (1992).

6. Cattell, R. G. G. *Object Data Management.* Englewood Cliffs, New Jersey: Prentice Hall, 1991.

7. Colley, Grant. "Retain the Object Model and Retain the Benefits," *Object Magazine* (May 1992).

8. Cossentine, Jay A. *Developing a Sophisticated User Interface to Support Domain-Oriented Application Composition and Generation Systems.* MS thesis, Air Force Institute of Technology, December 1993.

9. D'Ippolito, Richard and Kenneth Lee. "Modeling Software Systems by Domains." *Tenth Automating Software Design Workshop.* American Association for Artificial Intelligence, April 1992.

10. D'Ippolito, Richard S. "Using Models in Software Engineering." *Tri-Ada Conference.* 256–265. 1989.

11. Gool, Warren E. *Alternative Architectures for Domain-Oriented Application Composition and Generation Systems.* MS thesis, AFIT/GCS/ENG/93D-11, Air Force Institute of Technology, December 1993.

12. Halloran, Timothy J. *Performance Measurement of Three Commercial Object-Oriented Database Management Systems.* MS thesis, AFIT/GCS/ENG/93D-12, Air Force Institute of Technology, December 1993.

13. Harmon, Paul. "Object-Oriented Database Products, Part I," *Object-Oriented Strategies*, *III*(8) (August 1993). From Cutter Information Corp.

14. Intellectic International SNC. *Matisse 2.1.0 Object Editor User's Guide* (First Edition), June 1992.

15. Intellectic International SNC. *Matisse 2.1.0 Server Engine Administrator's Guide* (First Edition), June 1992.

16. Kang, Kyo C. and others. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990 (AD-A235 785).

17. Korth, Henry F. and Abraham Silberschatz. *Database System Concepts*. New York: McGraw-Hill, Inc, 1991.

18. Lamb, Charles, et al. "The ObjectStore Database System," *Communications of the ACM*, *34*(10) (October 1991).

19. Lee, Kenneth J. and others. *Model-Based Software Development (Draft)*. Technical Report CMU/SEI-92-SR-00, Software Engineering Institute, December 1991.

20. Lowry, Michael R. "Software Engineering in the Twenty-first Century." *Automating Software Design*, edited by Michael R. Lowry and Robert D. McCartney. 627–654. Menlo Park, CA: AAAI Press, 1991.

21. Mallare, Bradley and Mary Boom. *Formalization and Transformation of Informal Analysis Models Into Executable Refine Specifications*. MS thesis, AFIT/GCS/ENG/92-04, Air Force Institute of Technology, December 1992.

22. Mano, M. Morris. *Computer System Architecture*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1976.

23. Object Management Group. *The Common Object Request Broker: Architecture and Specification*. 492 Old Connecticut Path, Farmingham, MA 01701: Object Management Group and X/Open, 1991.

24. Object Management Group. *Object Management Architecture Guide*. 492 Old Connecticut Path, Farmingham, MA 01701: Object Management Group, 1992.

25. Ogush, Mike. "A Software Reuse Lexicon," *Cross Talk*, 41–45 (December 1992).

26. Randour, Mary Anne. *Creating and Manipulating a Domain-Specific Formal Object Base*. MS thesis, AFIT/GCS/ENG/92D-13, Air Force Institute of Technology, December 1992.

27. Reasoning Systems, Inc. *Refine User's Guide*. Reasoning Systems, Inc., Palo Alto, California, May 1990.

28. Roth, Major Mark A., "CSCE 646 - class title??." Class Notes, 1993.

29. Rumbaugh, James, et al. *Object-Oriented Modeling and Design*. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

30. Sandy, Raleigh A. III. *A Method for Populating the Knowledge Base of APTAS, A Domain-Oriented Application Composition System*. MS thesis, AFIT/GCE/ENG/93D-13, Air Force Institute of Technology, December 1993.

31. Stonebraker, Michael, et al. "Third Generation Data Base System Manifesto." *Proceedings of the IFIP DS-4 Workshop on Object-Oriented Databases*. July 1990.

32. Sun Microsystems, Inc., "Introduction to the ToolTalk Service." A White Paper, 1992.

33. Waggoner, Robert W. *Domain Modeling of Time-Dependent Systems*. MS thesis, AFIT/GCS/ENG/93D-23, Air Force Institute of Technology, December 1993.

34. Warner, Russell M. *A Method for Populating the Knowledge Base of AFIT's Domain-Oriented Application Composition System.* MS thesis, AFIT/GCS/ENG/93D-24, Air Force Institute of Technology, December 1993.

35. Weide, Timothy. *Development of a Visual System for a Domain-Oriented Application Composition System.* MS thesis, AFIT/GCS/ENG/93M-05, Air Force Institute of Technology, March 1993.

36. Welgan, Robert L. *Domain Analysis and Modeling of a Model-Based Software Executive.* MS thesis, AFIT/GCS/ENG/93D-25, Air Force Institute of Technology, December 1993.

Vita

Captain Danny A. Cecil was born on May 8, 1958 in Sidney, Ohio and graduated from Miami East High School in Casstown, Ohio in 1976. He enlisted in the Air Force in June 1976 and completed technical training for nuclear weapons maintenance at Lowry AFB, Colorado in December 1976. After one assignment as a nuclear weapons technician at Seymour Johnson AFB, North Carolina, he cross-trained into the computer programming career field. He spent four years as a programmer at Langley AFB, Virginia before being accepted into the Airmen Education and Commissioning Program. He attended Wright State University in Fairborn, Ohio from September 1984 to July 1987. Upon graduating cum laude with a Bachelor of Science in Computer Science degree, he attended Officer Training School and received his commission in October 1987. He attended the Basic Communications-Computer Systems course at Keesler AFB, Mississippi, graduating in February 1987. He was reassigned to Bergstrom AFB, Texas where he provided computer support for the Twelfth Air Force Air Component Commander's command and control commitments. In May 1992, Captain Cecil entered the Air Force Institute of Technology at Wright-Patterson AFB, Ohio to pursue a Master of Science degree with a concentration in software engineering.

Permanent address: 36 Regency Sq
Tipp City, Ohio 45371

Vita

Captain Joseph A. Fullenkamp was born on February 22, 1959 in Dayton, Ohio and graduated from Miamisburg High School in Miamisburg, Ohio in 1977. He enlisted in the Air Force in March 1978 and completed technical training for computer operations at Sheppard AFB, Texas in June 1978. After computer operations assignments at Dyess AFB, Texas and Langley AFB, Virginia, he was accepted into the Airmen Education and Commissioning Program and entered Wright State University at Fairborn, Ohio in June 1986. Upon graduating Summa cum laude with a Bachelor of Science in Computer Science degree in December 1987, he attended Officer Training School and received his commission in April 1988. As a new lieutenant, he attended the Basic Communications-Computer Systems course at Keesler AFB, Mississippi, graduating in September 1987. He was reassigned to Scott AFB, Illinois where he maintained the system software supporting Military Airlift Command's world-wide Consolidated Aerial Ports Subsystems. In May 1992, Captain Fullenkamp entered the Air Force Institute of Technology at Wright-Patterson AFB, Ohio to pursue a Master of Science degree in Computer Systems.

Permanent address: 2523 Sheridan Road
Omaha, Nebraska 68123

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
December 1993

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
Using Database Technology to Support Domain-Oriented Application Composition Systems

5. FUNDING NUMBERS

6. AUTHOR(S)
Capt Danny A. Cecil
Capt Joseph A. Fullenkamp

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/93D-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt Rick Painter
2241 Avionics Circle, Suite 16
WL/AAWA-1 BLD 620
Wright-Patterson AFB, OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research designed and prototyped an OODBMS technology base to store and retrieve various types of domain artifacts for domain-oriented application composition systems (DOACS). We developed object-oriented database schemas for a validating domain and the Object-Connection-Update software architecture. We implemented an inheritance relationship between the schemas so a domain model can inherit an architectural structure from an architecture model allowing us to isolate domain-specific knowledge from architecture-specific knowledge. We also developed a meta-model to formally define domain models in the database. We then developed a set of database methods to transform a domain model into a database schema for storing artifacts from the domain and to automatically populate the DOACS object base with the domain definition. Using an OODBMS, the structure and relationships that provide much of the power in object models are retained because the artificial flattening required for storage in traditional databases and file systems is prevented. Isolating domain and architecture models from each other has greatly increased the reusability of the domain artifacts, the domain model, and the architecture model. The inheritance relationship between domain and architecture models allows a domain to be defined once, but used in many different architectural environments and vice versa.

14. SUBJECT TERMS
computers, computer programs, software engineering, specifications, software architecture models, application composition systems, domain modeling, database, object-oriented database

15. NUMBER OF PAGES
223

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL